AUTOMATED MACHINE LEARNING FOR SUPERVISED AND UNSUPERVISED MODELS WITH ARTIFICIAL NEURAL NETWORKS

BY ALIREZA NAGHIZADEH

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Dimitris N. Metaxas

and approved by

New Brunswick, New Jersey

January, 2021

ABSTRACT OF THE DISSERTATION

AUTOMATED MACHINE LEARNING FOR SUPERVISED AND UNSUPERVISED MODELS WITH ARTIFICIAL NEURAL NETWORKS

by Alireza Naghizadeh

Dissertation Director: Dimitris N. Metaxas

Artificial Neural Networks (ANNs) are powerful machine learning tools to find and apply patterns for intelligent decision making. These tools can be combined with automation to select few results among many trials. Since ANNs are used for both supervised and unsupervised learning, automation can lead to more trusted learning methods across many fields and lead to exploring possibilities that are considered impossible with current technology. In this thesis, at first, I introduce a new form of ANN architecture which is used exclusively for automated robot navigation. By doing so, I provide a high-level overview of both computational neuroscience and the potential of automation. Next, I introduce Greedy AutoAugment to automate the learning of state-of-the-art neural networks for both big and small datasets. I also create an efficient model to evaluate clustering in unsupervised learning. The model is further expanded to introduce unsupervised learning for deep subspace clustering. In the end, I provide discussion and the future research plan for automating ANNs in machine learning applications.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Prof. Dimitris N. Metaxas, for his support and guidance throughout the course of my Ph.D. I would also like to thank the other members of my Ph.D. committee, Profs. Ahmed Elgammal, Abdeslam Boularias, and Wei Vivian Li for reviewing this thesis. Additionally, I would like to thank Prof. David J. Margolis for mentoring and teaching me about the field of neuroscience. I want to thank all of my other collaborators, teachers, and mentors who I've had the pleasure of learning from throughout my Ph.D. studies. Finally, I'd like to thank all of the members of my family who have encouraged and supported me over the course of this Ph.D.

Dedication

”And he has bestowed upon you all that you asked for; And were you to count the blessings of Allah, you would not be able to list them.” Q. 14:34.

To my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

The purpose of machine learning is to find and use the patterns in data for making rational and grounded decisions. There are two major paradigms for pattern discovery in data, which are called supervised and unsupervised learning. In supervised learning, a training set is present, so patterns of the data can be found from the train set and then be generalized for the test set. In unsupervised learning, a train set is not present. Therefore, it is more challenging to discover patterns on data. In machine learning, there are several types of learning forms, such as finding the distribution of training data points and generating similar data, finding objects, and performing semantic segmentation. The most basic form of machine learning, which will be used in this thesis, is to categorize multivariate points into different groups. In supervised learning, this process is called classification, and in unsupervised learning, it is called clustering.

Artificial Neural Networks (ANNs) are computation models of neural activities in the brain of animals and human beings. In general, the study of finding a relationship between the activities of the outside world and their corresponding neural activities (macro to the micro) is a difficult task, which usually does not lead to reproducible results. Therefore ANNs are mostly inspirational and do not necessarily follow the real process of biological evidence. The architectures of the neural networks usually consist of two main elements. The first element is the computational neural cells, which take numerical values that have major effects on the decision making process. The second element is the connections between computational neural cells. These connections can also take numerical values, which lead to the final calculations of computational neural cells. These two basic components can be structured into complex models where different forms of learning can take place.

In recent years, the success of ANNs for breaking different types of benchmark challenges, convenient access to application programming interfaces on GPUs, and computer vision libraries led to the exponential growth of these types of learning methods. The ability to solve complex problems with neural networks helped to create smarter applications for many fields such as virtual assistants, search engines, medical applications, media, and image editing. In recent years, there has been a trend to combine automation with ANNs. The automation, in this case, is a general term that includes any technology that reduces human interventions in complex learning systems. If automation is combined with ANNs, it can provide numerous advantages. Some of these advantages are 1- it can prevent common human errors. 2- In some cases, the requirement for experts in the field can be reduced hugely. 3- It saves time, and 4- it saves resources. Another important result for such a combination, which is the main focus of this thesis, is to use automation to help the learning itself. Machines can explore many possibilities compared to humans. If we define a criterion for identifying the best trials, we will see that they can be used to improve the learning ability of the networks.

1.2 Contributions of the Dissertation

The contributions of the thesis are in three major chapters. In the first chapter, we design a simple automated neural network architecture by using patterns of the brain. By doing so, we provide a high-level overview of both computational neuroscience and the potential of automation. In the second chapter, we automate the learning process in state-of-the-art supervised networks for both small and big datasets. In the third chapter, we explore important aspects of unsupervised learning, which are related to automation. We propose solutions to automate both high-level and low-level clustering algorithms.

1.2.1 Automated Artificial Neural Networks: An Example

In the first stage of the thesis, we show how to design a new form of ANN architecture by getting inspiration from neural activities in the brain. There are also elements of automation in the method that can help readers to get familiar with both concepts. For the neural activities, we use grid cell firing patterns in the medial entorhinal cortex, which is shown that it can be used as a mapping reference for spatial navigation in mice and other mammalian species. We use this pattern to propose a novel computational model for patterns of grid cells and combine it with a mechanism to tune the weights of cells, which we use to create a decision-making process for robot navigation [10]. The method is automated to be used as an unsupervised method for uninformed online search with unknown goal positions and unknown environments, such as finding the exit out of a maze or help a robot to find its way in a jungle where there is no clue about the exit.

The results of this approach in simulated and real environments show superior algorithmic steps over current search methods. In addition, the typical size of the memory can be reduced without compromising the completeness of the method. Our results show that the number of steps is stable in terms of variations in memory allocations. While this method is effective for the specific task that it is designed for, it cannot be generalized easily to other forms of learning. For general learning models, we use more advanced ANN architectures. These networks are designed to work on multivariate data points, and instead of focusing on a specific task, they provide a more general form of the learning model that can be used in a variety of tasks.

1.2.2 Automated Supervised Learning

In pattern recognition, we want the perception of an object to be invariant to the properties that can vary in different environments such as scale, brightness, rotation, and viewing angle. For instance, it is desirable that a network, after learning an object from its original form, recognizes the same object with a change of location or added rotation. Currently, there are two ways to deal with this problem. First, by designing network architectures that inherently can be invariant to important image transformations. Second, with data augmentations. The most basic network which considers the transformations of the input data is the Convolutional Neural Network (CNN). The CNN architecture, with the concept of convolutional layers, tries to be translation invariant. This network was very successful in its approach and has been used as a base for the development of more advanced architectures.

The second method for considering different transformations of the input data is to use data augmentation. In this method, we want to achieve invariance by applying various transformations on training data. For instance, both the original data and a rotated transformation of the data are used to train the network. The main advantage of this method is simplicity and supporting all forms of deep network architectures. Because of the importance of this problem and the impact that this approach can have on ANNs, for the rest of the thesis, we focus on automating this form of learning.

A major problem in automating data augmentation is to ensure that the generated new samples cover the search space. This is a challenging problem and requires exploration for data augmentation policies to ensure their effectiveness in covering the search space. To solve this problem, we propose Greedy AutoAugment [11] as a highly efficient search algorithm to find the best augmentation policies. We use a greedy approach to reduce the exponential growth of the number of possible trials to linear growth. The Greedy Search also helps us to lead the search towards the sub-policies with better results, which eventually helps to increase the accuracy. Our experiments on four datasets (Tiny ImageNet, CIFAR-10, CIFAR-100, and SVHN) show that Greedy AutoAugment provides better accuracy, while using 360 times fewer computational resources.

Next to providing automation for augmenting normal datasets, we also provide augmentation for n-shot learning [12]. The goal of n-shot learning is the classification of input data from small datasets. This type of learning is challenging in neural networks, which typically need a high number of data during the training process. Greedy Autoaugment can produce an infinite number of target conditions from the primary condition. This process includes two main steps for finding the best augmentations and training the data with the new augmentation techniques. In Chapter 3.2, we optimize these two steps for n-shot learning. The proposed method can potentially extract many possible types of information from a small number of available data points in n-shot learning. The results of our experiments on five prominent n-shot learning datasets show the effectiveness of the proposed method.

1.2.3 Automated Unsupervised Learning

To automate unsupervised learning for clustering, it is important to know about the presupposition of patterns in the clustering. To better underestand this idea, in Chapter 4.1 a new form of pattern for clustering is introduced. For the pattern, we assume the decomposition of points with the mixture of Gaussian distributions in each dimension as an underlying assumption for feature formation of input data. The new guideline presents a unified approach to current basic assumptions and also provides us with an opportunity to solve an essential problem of low-level clustering algorithms. The issue is in the form of the curse of dimensionality, which claims that multivariate clustering is meaningless for high dimensional data. To solve this problem, we propose a new type of vector norm ($\|.\|_c$) and subsequently, Clustering Distance (CD), which is a distance metric system that guarantees meaningfulness even in high dimensional data [13]. The experiments on synthetic and non-synthetic datasets show the effectiveness of the proposed method to solve curse of dimensionality.

While the new pattern has nice features and can solve some problems in clustering, it is still easier and better to work with current presupposed patterns. To automate the learning process for clustering, the next step is to understand the evaluation metrics. For clustering methods, K-Means is an important algorithm that is used as a helper function for many other algorithms. Therefore, to properly automate the unsupervised learning, it is important to have an evaluation metric for K-Means. In K-Means based clustering algorithms, different initial seeds can lead to different clustering results. Selecting the best result from different initial seeds is called the filtering process and will be used for automation.

The filtering process follows three steps, 1- performing several clustering trials, 2- scoring each trial, and 3- choosing the trial with the best score. A typical method to score the clustering results of K-Means is the within-cluster sum of squares (WCSS). There are more advanced methods that can be used to score the clustering trials. These methods usually provide a better score with the cost of being more computationally demanding. Before automating the clustering process with Greedy Autoaugment, we fix this problem by proposing Condensed Silhouette [14], which is a very efficient version of the Silhouette algorithm. For this purpose, we replace the elements of Silhouette algorithm with similar elements of the K-Means algorithm. This helps us to maintain the accuracy of the Silhouette and, at the same time, significantly reduce the computational requirements of the method. Our experiments on 14 real datasets show the effectiveness of the proposed method.

The above process automates clustering for low-level clustering algorithms. To cluster complex data such as images, we use high-level clustering algorithms such as deep subspace clustering (DSC). As we mentioned, the idea behind data augmentation techniques is based on the fact that slight changes in the percept do not change the brain cognition. In classification, neural networks use this fact by applying transformations to the inputs to learn to predict the same label. However, in DSC, the ground-truth labels are not available, and as a result, one cannot easily use data augmentation techniques. We propose a technique to exploit the benefits of data augmentation in DSC algorithms [15, 16]. We learn representations that have consistent subspaces for slightly transformed inputs.

In particular, we introduce a temporal ensembling component to the objective function of DSC algorithms to enable the DSC networks to maintain consistent subspaces for random transformations in the input data. In addition, we provide a simple yet effective unsupervised procedure to find efficient data augmentation policies. We search through the policies in a search space of the most common augmentation policies to find the best policy such that the DSC network yields the highest mean Silhouette coefficient in its clustering results on a target dataset. Our method achieves state-of-the-art performance on four standard subspace clustering datasets.

1.3 Outline of the Dissertation

The remainder of this thesis follows this outline:

- In Chapter 2, we create a new form of ANN by observing the patterns of Grid Cells. This chapter shows to the readers that how learning can take place with neural nets and how the computational field can connect to the field of Neuroscience. The design also demonstrates the strength of automation in learning.

- In Chapter 3, we present the Greedy AutoAuggment to automate and improve the learning process of ANNs by using different representations of images.

- Chapter 3.1 is for normal datasets and Chapter 3.2 is for n-shot learning (small datasets).

- In Chapter 4, we present automation for unsupervised learning.

- Chapter 4.1, is to show the importance of pre-defined patterns in clustering and their effects on final results.

- Chapter 4.2, uses the pre-defined pattern of K-Means and makes Silhouette feasible for this algorithm even for high number of points.

- Chapter 4.3, uses the knowledge of Chapter 4.1 and 4.2 to define Greedy AutoAugmentation for unsupervised learning.

Chapter 2

Automated Artificial Neural Networks: An Example

2.1 Introduction

Grid cells, which are located in the medial entorhinal cortex (MEC), are sets of neurons that illustrate periodic patterns when an animal traverses a region [17, 18, 19] and have been found in bats, mice and human beings [17, 20, 21, 22]. The cells represent a unique firing pattern which led to a hypothesis that they make a cognitive structure of Euclidean Space [17, 23] and exhibit a hexagonal lattice that covers the region. The discovery and characterization of grid cells and place cells of the hippocampus led to the 2014 Nobel prize in Physiology or Medicine [24].

Grid cells have modular organization [25, 26] consisting of groups of interconnected neurons. One of the outstanding features of grid cells is their firing fields are mostly related to distance from the origin rather than the inputs of visual cues [27, 28, 29]. These features, along with the triangular pattern of grid cells, make them a suitable metric of the brain's spatial representation system [30, 31]. Also, the firing patterns represent codes of the location, which are similar to a mathematical residue system that can be used as a utility for metric navigation [32, 33]. Since the discovery of grid cells, several studies have provided computational models that satisfy the lattice layout of grid cells [34, 35, 36]. A search model based on grid cells [1] can contribute to the robotic navigational algorithms. In these algorithms, we are mostly concerned about finding the shortest paths, reducing the computational complexity and using the lowest possible memory usage.

In robot navigation, an underlying infrastructure such as Simultaneous Localization and Mapping (SLAM) [37] is usually used to provide 1) topological mapping of the environment, 2)

[1]Throughout this section, the keyword ”grid cell” is used for its biological definition. This definition is different from some engineering literature which may use this keyword for other porpuses.

self-awareness of the current positions for robots, 3) decision-making structure for exploration. The focus of this section is the third part which is path-planning for robotic exploration. More specifically, we are concerned with exploring unknown environments to decide how to get from a start location to an end location without knowing the position of the end location. It is desired to reach the end location with a small number of steps [2] and memory usage. In this case, in every turn a robot can select between a limited number of options, (up, down, left, right, up-right, up-left, down-right, down-left) which is usually up to eight moves in a grid topological map. Therefore, the computational complexity for all of the algorithms in each turn is O(1) and is not important.

There is already an enormous body of literature on robotic navigation, including navigation in unknown environments. There are even some early attempts such as ALVINN [38] to use neural networks in autonomous vehicles. A large number of path-planning algorithms can be used in combination with SLAM. Some of these algorithms restrict the environment or require some assumptions to simplify the problem. For instance, the heuristic based algorithms such as A*, D* and their siblings [39, 40, 41, 42, 43] require some forms of information about the exit which is also true for bugs family algorithms [44, 45]. In this regard, while unknown environments are presumed, the position of the goal should be known a priori. Also, the Wall-follower and Pledge algorithms [46, 47] need specific conditions for the environments to preserve the completeness. Some methods such as sampling-based planners (e.g., RRT) [46, 47] also solve the problems but typically need full disclosure of the environment or at least the position of the goal [48]. Current efforts which use grid-cells or other forms of neural networks for path-planning [49, 50, 51] also need the position of the goal or follow a supervised scheme where training is required before exploration [52].

The breadth-first search (BFS) [53, 54] and depth-first search (DFS) with backtracking feature [55, 56, 57] seem to be the first reasonable answers for path-planning exploration, to find the exit in unknown places where a physical agent does not have any information about the environment or the position of the goal. The BFS in this regard is similar to Iterative Deepening Depth-First Search (IDDFS) algorithm [58]. The DFS with backtracking feature is

[2]Throughout the section, we use "algorithmic steps" and "speed" interchangeably to indicate a number of the steps from start location to the end location.

usually called Backtrack algorithm. A better approach is to use LRTA* [59] with a constant heuristic value such as zero. While not very informed, this satisfies the constraint of admissible heuristic and therefore provides us with a suitable uninformed algorithm for comparisons [60]. It should be mentioned that current improvements of LRTA*[61, 62, 63], to the best of our knowledge rely on the heuristic aspects of the algorithm (e.g., goal's position) and make the comparisons impossible.

The last possibility is to use Markov Decision Processes (MDPs) which directs navigation by setting policies for all possible actions of all possible states. The generic MDP process needs full disclosure of the environment to set best policies which are not applicable in unknown environments. That is why we have to use reinforcement learning in the context of MDP to learn policies with explorations [64]. Among several options, we use Q-Learning which is still used for robot exploration [65, 66] next to other areas such as deep reinforcement learning for solving games [67]. However, as we see in the results, since finding the best policies is a slow process, these algorithms are not the best options where the purpose is to find a specific location which limits the exploration aspect of finding policies. As a final note, ant colony optimization (ACO) [68] is capable of solving the problem. However, since ACO techniques are intrinsically multi-agent, they are not compatible with the proposed method.

In this section, we propose the Gridcell Navigational Model (GNM) to automate a general robotic search solution in entirely unknown environments and unknown goal positions. The search algorithm can be interpreted as a task-specific form of Artificial Neural Network (ANN) which basically amounts to the random walk with a reduced probability to visit previously visited states. For doing this, first, we propose a grid cell computational model which is suitable for a robotic agent. This model provides us with an effective mapping representation of the 2D map that we receive from the SLAM infrastructure. Second, we jointly use the computational model and a weight tuning mechanism to create an unsupervised learning assistant. If we consider the working environment as a Graph, the learning of the proposed method is focused on the nodes of the graph. This design is in contrast to the methods such as Q-Learning which focus on learning the links of the graphs. Since the number of the nodes in complex environments is usually significantly smaller than the number of the links, the payoff is also great where we can learn and avoid unwanted regions quickly.

With the new neural representation, there is also a motivation to encode the environment in fewer states. Consequently, the memory of GNM has a dynamic nature where we can use smaller than ideal memory space which is a trade-off with algorithmic steps of the agent. This attempt to create an effective search model with a grid representation of the topological map is different from map refinement methods such as [69, 70, 71] where their main objective is to create more accurate maps. Moreover, unlike frontier-based algorithms[72], we do not assume the detection of the end goal from far away. In this regard, we address the problem in the most general way without assumptions about the type/range of the sensors. The same assumption is taken from other navigational algorithms mentioned previously. If such sensors were avaliable, it would be possible to incorporate recognition of the target first or in the middle of the way and then finding the solution would be much easier. Without such assumption, the frontier-based algorithms are either incomplete or, with some modifications reduce to the more general cases such as Backtrack.

While biological evidence inspires this process, our purpose is to provide a reliable path-finding algorithm which may not necessarily reflect the real process of biological navigation. Our experiments show superior results on algorithmic steps. In the cases where we do not limit the memory, the memory usage is still competitive. We show that when memory is limited, the method provides better algorithmic steps than IDDFS which has the best memory usage. We demonstrate that even with smallest memory, the method converges to the solution. The GNM can be used as an effective method for robot navigation. In the future, the method can be combined with other path-planning solutions such as map refinement methods or frontier-based algorithms. Similar to other path-planing algorithms, the GNM can potentially be introduced as a practical solution for solving problems in other fields that have compatible search settings such as routing in computer networks [73, 74, 75, 76].

2.2 GridCell Computational Model

The computational model of grid cells is the corresponding interactions of neurons in the grid cell module, given the movements of an agent in an environment. In this regard, we present the computational model of grid cells by defining three main components, 1) environment 2)

Rule.No	Actions	b.status
1	Start search	Start
2	Can dispatch	Open
3	Can dispatch	Visited
4	Cannot dispatch	Close
5	End search	End

Table 2.1: List of possible actions.

grid cell module 3) the interaction between environment and the grid cell module.

Environment - The SLAM infrastructures can provide an abstract and isolated representation of the real world. For this purpose, robots consider the smoothness of the path with the constraints of speed, acceleration, the radius of turning circle and other criteria. The acquired topological map can be transformed into a grid representation (for more information see Section 5.1). Let's define each grid as a two-dimensional lattice, **E** which provides a transformed environment for a mobile agent, **A**. In this regard, the transformed environment has a higher abstraction of the real world and is enough to help for the movements of **A** without worrying about details of SLAM protocols.

Each transformed environment **E**, consists of several blocks $b_{ij}, i = 0, \dots x-1, j = 0, \dots, y-1$ in which the agent can roam freely. We consider block b as a square shape with sides that are at least long enough to contain the **A** inside its borders. The blocks hold important information about their status which is required to provide a searching environment for the agent. In summary, these environments are standard lattices where all transitions from one state into another state have the same value. The status of b, ($b.status$) provides a precondition for the executability of an action and can take one of the forms as *open*, *close*, *visited*, *start* and *end*. Accordingly, the acceptable actions of **A** are based on Table2.1.

Grid cell module - According to the biological evidence, grid cells are packed into modules which are distributed into entorhinal cortex of medial temporal lobe [25, 26]. In this design, we also use a similar paradigm, but in practice, only one module is allocated. Let us define a two-dimensional lattice, **M** which represents a module that contains all of the required grid cells. This module can be considered as an abstract and isolated representation of brain for **A** to help it for navigation in **E**. The **M** consists of several cells $c_{ij}, i = 0, \dots m-1, j = 0, \dots, n-1$ which represent the same behavior of grid cells in brain. From here, the maximum number of cells in $\mathbf{M}$ is equal to $m \times n$. The size of the $\mathbf{M}$ is based on a parameter h, which also indicates the firing locations for each cell. This process is defined in the computational model.

Computational model - In our navigational system, for each block b of the $\mathbf{E}$, there is a cell c in $\mathbf{M}$ that fires. At this point, based on the movements of $\mathbf{A}$, different forms of patterns can be created in $\mathbf{E}$. However, to follow the biological representation of grid cells in 2D space, the computational model requires the creation of triangular firing patterns [77]. Therefore, our goal is to fire c when agent resides in the edges of triangles (in $\mathbf{E}$) which are responsible for firing c. We denote M_{d_r} as the horizontal distance from the beginning of the row and M_{d_c} as the vertical distance from the beginning of the column of the module $\mathbf{M}$.

As mentioned, for each cell c (in $\mathbf{M}$), it fires if $\mathbf{A}$ is in either one of three edges of the triangle that are responsible for triggering that particular cell. To determine triangular edges in $\mathbf{E}$, we use dimensions of $\mathbf{M}$ and denote the maximum size of M_{d_r} as $m = 2h$ and the maximum size of M_{d_c} as $n = 2h - 1$ where h is any positive integer number. To create the triangular pattern, we set the size of the first side of the triangle in $\mathbf{E}$ for each c to be $2h + 1$. In this way, the first two edges can easily be created from the horizontal perspective. The third edge is created with a vertical distance of $2h - 1$ and horizontal distance of h. All of the distances are relative to the position of the agent in the $\mathbf{E}$ at the beginning of the search. This helps us to set the agent in any block of the $\mathbf{E}$ for starting a search.

As an example, consider Figure 4.1. The mouse (agent), starts its search from s with position (0,0) and wants to reach e in environment $\mathbf{E}$. To this end, it uses module $\mathbf{M}$ which is constructed based on $h = 2$. The value of $h = 2$ gives $m = 2h = 4$ and $n = 2h - 1 = 3$. All of the blocks of $\mathbf{E}$ have to be covered by cells of $\mathbf{M}$. We have marked two cells in $\mathbf{M}$ with pink and red colors and represented exactly where in $\mathbf{E}$ they would fire. We also connected the blocks with pink colors of $\mathbf{E}$ to show the triangular pattern of the cell with the pink mark. Each cell in $\mathbf{M}$ creates a similar triangular pattern in the environment.

Based on the process as mentioned above, we present the firing pattern of each cell. The purpose is to determine that when $\mathbf{A}$ takes horizontal and vertical distances from the position s, which cells in $\mathbf{M}$ should fire. We denote $\mathbf{E}_{d_r}$ as the horizontal distance of $\mathbf{A}$ from position of the s and $\mathbf{E}_{d_c}$ as the vertical distance from the position of the s. Accordingly, we present

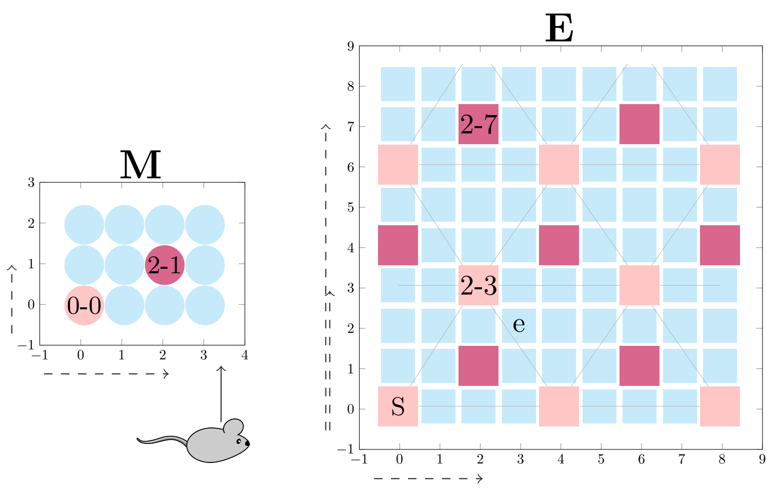


Figure 2.1: Each cell of **M** is responsible for firing in specific triangular edges of **E**. The change of location in **E** makes cells in **M** to fire. In this example, two cells in **M** are highlighted and it is shown where in **E** they would fire.

two equations. If the lower bound of $\frac{\mathbf{E}_{d_c}}{n}$ is an even number (e.g., $\left\lfloor \frac{\mathbf{E}_{d_c}}{n} \right\rfloor = 2k$). We have,

$$\begin{cases} \mathbf{M}_{d_r} = \mathbf{E}_{d_r} \; mod \, m \\ \mathbf{M}_{d_c} = \mathbf{E}_{d_c} \; mod \, n \\ fire(c, \mathbf{M}_{d_r}, \mathbf{M}_{d_c}) \end{cases} \tag{2.1}$$

in which, mod is the modulo operation and m, n indicate maximum lengths of row and column of **M**. Subsequently, if the lower bound of $\frac{E_{d_c}}{n}$ is an odd number (e.g., $\left\lfloor \frac{E_{d_c}}{n} \right\rfloor = 2k+1$), we have,

$$\begin{cases} \mathbf{M}_{d_r} = (\mathbf{E}_{d_r} + h) \; mod \, m \\ \mathbf{M}_{d_c} = \mathbf{E}_{d_c} \; mod \, n \\ fire(c, \mathbf{M}_{d_r}, \mathbf{M}_{d_c}) \end{cases} \tag{2.2}$$

In the above equations, the function $fire(c, \mathbf{M}_{d_r}, \mathbf{M}_{d_c})$ fires the cell c in **M** with locations of M_{d_r} and M_{d_c} distances.

To clarify this process, observe examples of blocks b_{23} and b_{27} in Figure 4.1. Let's determine the cell that is responsible for firing, when **A** visits b_{23}. Considering (2.1),(2.2), we have, $\left\lfloor \frac{E_{d_c}}{n} \right\rfloor = \left\lfloor \frac{3}{3} \right\rfloor = 1$ which is an odd number so we use (2.2). We have,

$$\begin{cases} \mathbf{M}_{d_r} = (2+2)\ \ mod\ \ 4 = 0 \\ \mathbf{M}_{d_c} = 3\, mod\, 3 = 0 \\ fire(c, 0, 0) \end{cases}$$

Accordingly, the cell c_{00} fires. By following the same trend, for b_{27}, we have $\lfloor \frac{7}{3} \rfloor = 2$ and use (2.1), therefore, the cell c_{21} fires.

$$\begin{cases} \mathbf{M}_{d_r} = 2\, mod\, 4 = 2 \\ \mathbf{M}_{d_c} = 7\, mod\, 3 = 1 \\ fire(c, 2, 1) \end{cases}$$

This example shows how the computational model works. As we can see, in every block of the environment, it is specified which cell in the module fires. However, this information alone is not going to help the agent to find its way to the end block. For this purpose, we need a decision-making structure over the firing patterns of the neurons.

2.3 Decision-Making Structure for Movements

To create our search model, the computational model is combined with a decision-making structure in two different ways. 1) the ”GNM Basic,” which is a preliminary version that is used for navigation and finding the exit. 2) the complete GNM algorithm which also can discover closed paths and improves the decision-making of the agent. We use the term ”closed path” to describe a scenario with no possibility of reaching the goal state.

2.3.1 GNM Basic

Based on the Hebbian learning rule [78], if the firing pattern of one neuron leads to the firing of another neuron, synaptic contacts are strengthened between these neurons. We can use this biological premise to guide the search forward. In this regard, we follow a modified version of the Hebbian rule where instead of strengthening the connections between two fired neurons, the weights of the fired neurons are increased. The Hebbian rule, next to having plausibility in computational neuroscience, helps us to use Oja’s rule, which leads to the boundary of (0,1). Modifying the Hebbian rule to change the weights of cells instead of the connections, helps us to learn from the nodes instead of connections. The number of nodes in complex environments is usually significantly smaller than the number of links. This rule also applies to the defined environment in the previous section. Therefore, by focusing on the nodes instead of the links, we can learn and avoid unwanted regions quickly.

The goal is for agent $\mathbf{A}$ to go from start block to the end block. To reach the destination, $\mathbf{A}$ should prevent those blocks that it has visited before unless that block is required to discover unvisited blocks. We denote $c.w$ as the corresponding weight for cell c. Let's consider the movement of $\mathbf{A}$ from $b^{(1)}$ as the first block to $b^{(2)}$ as the second block. If $b^{(1)}$ is responsible for firing $c^{(1)}$ and if $b^{(2)}$ is responsible for firing $c^{(2)}$, the cells $c^{(1)}$ and $c^{(2)}$ fire together. Consequently, both $c^{(1)}.w$ and $c^{(2)}.w$ are increased when $\mathbf{A}$ goes from $b^{(1)}$ to $b^{(2)}$. Based on the above description, we present the weight tuning rule as follows,

$$\begin{aligned} c^{(1)}.w(t+1) &= c^{(1)}.w(t) + \eta c^{(1)}.w(t)c^{(2)}.w(t) \\ c^{(2)}.w(t+1) &= c^{(2)}.w(t) + \eta c^{(1)}.w(t)c^{(2)}.w(t) \end{aligned} \tag{2.3}$$

in which η is the learning rate. Considering $c^{(1)}, c^{(2)}$ are two grid cells that fire for $b^{(1)}, b^{(2)}$ when agent moves between two blocks, the $(t+1)$ represents the next state of the variables.

For choosing between the next destination among its neighbors, $\mathbf{A}$ simply selects nodes with lowest $c.w$, and in case of the equal weights, the equal probability of selection is given to nominees (random tie-breaking). In this regard, $\mathbf{A}$ prevents being trapped into the loops. It also avoids directions that previously did not give the satisfactory results but remains open to choosing them in the future if the current direction gains more weights than its counterparts. An illustration of this process is represented in Figure 4.4.

The search starts with b_{00}, in which we have $c_{00}.w = 0$. In step 1, $\mathbf{A}$ dispatches to b_{10}, we have $c_{00}.w = 1, c_{10}.w = 1$. In step 2, $\mathbf{A}$ dispatches to b_{20}, we have $c_{00}.w = 1, c_{10}.w = 2, c_{20}.w = 1$. In step 3, $\mathbf{A}$ dispatches to b_{21}, we have $c_{00}.w = 1, c_{10}.w = 2, c_{20}.w = 2, c_{21}.w = 1$.In this step, let's analyze the choices of $\mathbf{A}$ among blocks, $b_{11}, b_{22}, b_{31}, b_{20}$. Since $c_{20}.w > c_{11}.w, c_{22}.w, c_{31}.w$, the b_{20} has the least priority and is not considered. Since $c_{11}.w = c_{22}.w = c_{31}.w$, then b_{11}, b_{22}, b_{31} have the same priority and one of them is chosen randomly. Similarly, the search continues until agent reaches the end block. It should be noted

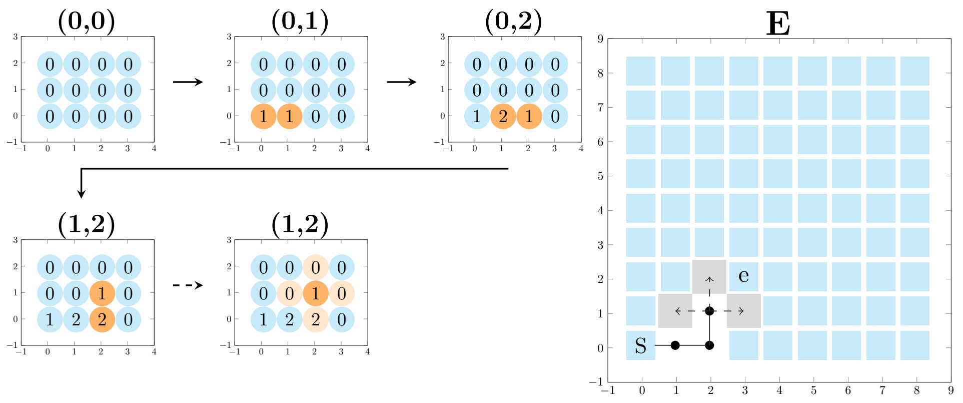


Figure 2.2: Example of a navigational process using Hebbian learning rule. Moving between two blocks strengthens their corresponding neurons.

that here we began the search with $c_{00}.w = 0$, but it is recommended to initialize the beginning node with η (here the value of 1) to count the starting node as once visited before the search starts.

2.3.2 GNM

In an environment, all of the paths to the destination may be closed. In this case, $\mathbf{A}$ increases the weights of the cells infinitely. Intuitively, an intelligent agent should finish the search when there is no possibility of finding new ways. Accordingly, to solve this problem for $\mathbf{A}$, the agent should remember all the blocks of $b.status = open$. In the next, we extend the GNM Basic to address this issue and further improve the search mechanism. To use the memory space more efficiently, we only use one variable for both marking the viewed states and storing their weights.

We expand the current weights of the cells as real numbers ($\mathbb{R}$) with two parts as $c.w.(p_1|p_2)$. In $c.w.(p_1|p_2)$, the p_1 stands for the left side and p_2 stands for the right side of the decimal. Each cell $c.w$, can have either p_1 or p_2 or both of them. Considering this fact, (2.3) is exclusively used to increase $c.w.p_1$. As a result, η is only defined as a natural number ($\mathbb{N}$).

We also define $\mathbf{A}.v$ as the viewing distance for agent $\mathbf{A}$. It determines that in what distance, $\mathbf{A}$ can view deeper into its neighbors. For instance, if $\mathbf{A}$ is in the block b_{ij}, with $\mathbf{A}.v = 1$, $\mathbf{A}$ can see $b_{(i+1)(j)}$, $b_{(i-1)(j)}$,$b_{(i)(j+1)}$ and $b_{(i)(j-1)}$. Similar to the search mechanism, we use

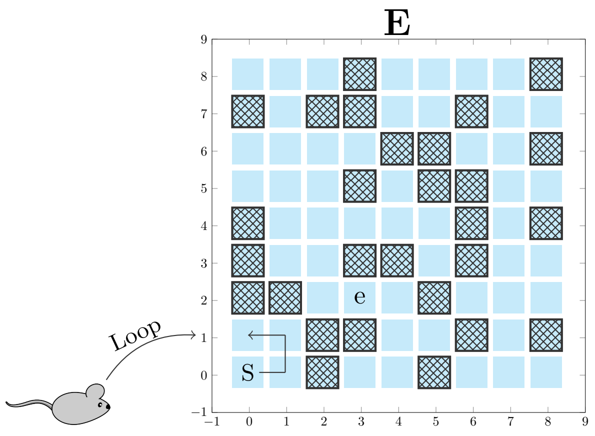


Figure 2.3: The closed blocks are represented by cross-hatch patterns. Agent adds all of the open blocks and in each visit of an open block reduces the value by 1.

weight tuning for viewing neighbors as well. In this case, because we are only allowed to work with the right side of the decimal number, we determine a boundary for $c.w.p_2$ which is $p_2 \in \Big(0, \quad 1\Big)$. To remain in this boundary, we use Oja's rule [79] as a simple normalization factor for weight tuning,

$$\begin{aligned}
&c^{(1)}.\mathrm{w}.p_2(t+1) = c^{(1)}.\mathrm{w}.p_2(t)+\\
&\eta_2\, c^{(2)}.\mathrm{w}.p_2(t)(c^{(1)}.\mathrm{w}.p_2(t) - c^{(1)}.w.p_2(t) \times c^{(2)}.\mathrm{w}.p_2(t))\\
&c^{(2)}.\mathrm{w}.p_2(t+1) = c^{(2)}.\mathrm{w}.p_2(t)+\\
&\eta_2\, c^{(1)}.\mathrm{w}.p_2(t)(c^{(2)}.\mathrm{w}.p_2(t) - c^{(2)}.w.p_2(t) \times c^{(1)}.\mathrm{w}.p_2(t))
\end{aligned} \tag{2.4}$$

Similar to (2.3), η_2 is the learning rate for $c.w.p_2$ and is a decimal number (η is used for $c.w.p_1$). The $(t+1)$ represents the next state of the variables and $c^{(1)}, c^{(2)}$ are two grid cells that fire for viewing from $b^{(1)}$ to $b^{(2)}$. Considering (2.4), the more views of c results to smaller values for $c.w.p_2$. The important factor in this equation is the initialization of $c.w.p_2$ which should follow the boundary of $p_2 \in \Big(0 \quad 1\Big)$.

To solve the problem of the closed path, we define $\mathbf{A}.counter$ which keeps track of the possibility of blocks as $b.status = open$. When $c.w.p_2$ is equal to the initialized value and $b.status = open$, the agent $\mathbf{A}$, by considering distance view $\mathbf{A}.v$, increases $\mathbf{A}.counter$ by one unit. When the agent moves to a block of $b.status = open$, it decreases $\mathbf{A}.counter$ by one

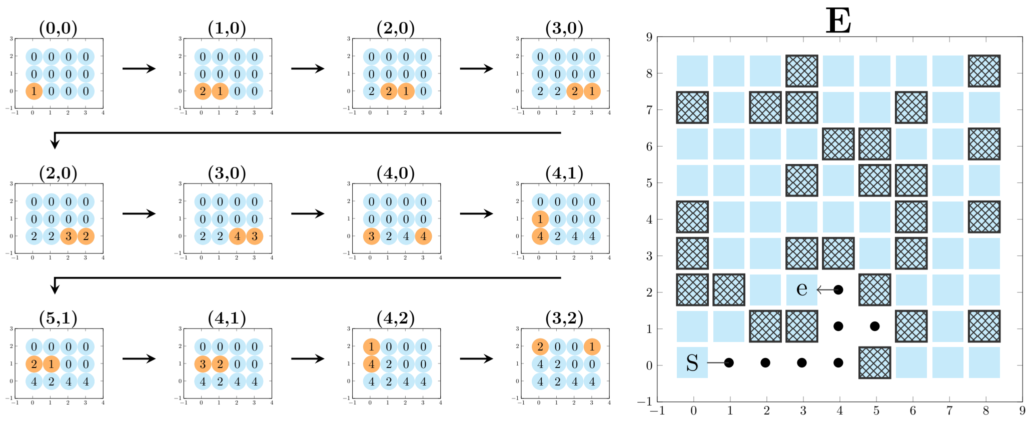


Figure 2.4: A complete navigational example of GNM in the presence of closed blocks for seeing the effects of smaller modules.

unit. When $\mathbf{A}.counter = 0$, the agent should finish the search since there is no possibility of finding new blocks. Therefore, as long as there is a viewed but un-visited block, the search continues.

Figure 4.2 continues the previous example where the agent wants to go from s to e. However, this time there is no way to find an open path from s to e. The first block is the starting block, so it is marked as the viewed, but it does not increase $\mathbf{A}.counter$. In the next step, b_{10}, b_{01} are marked as viewed and $\mathbf{A}.counter$ increases by 2. The agent goes to b_{10}, decreases the counter and sees that the only unvisited block is b_{11}. It marks the block as viewed, increases $\mathbf{A}.counter$ and goes to the b_{11} and decreases $\mathbf{A}.counter$. The only remaining place is b_{01}, but it is already marked as viewed so $\mathbf{A}.counter$ is not increased. $\mathbf{A}$ goes to the b_{01} and decreases $\mathbf{A}.counter$. At this stage, since $\mathbf{A}.counter = 0$ the search is stopped.

In conclusion, note that in case of multiple environments, since the proposed method is intrinsically unsupervised, we expect that different modules to be used for different environments. We can also reset the values of the module which we are not going to use immediately. Therefore, while using the same values in similar environments probably leads to better results, such improvements are not guaranteed especially in our search setting where environments are completely unknown. Even though the basic design of the search mechanism is complete, we can still further improve the search without overhead on the current infrastructure. In each step, the agent should decide between its neighbors. We define $\mathbf{S}$, as the set of all the neighbors that $\mathbf{A}$ can choose for its next move in each block b. In the first step, we give priorities to $c.w.p_1$ in which we only preserve those blocks with smallest p_1. If $\mathbf{S}$ has only one member, the choice is finished. Otherwise, we use $c.w.p_2$ and only preserve those blocks with highest p_2. Consequently, the agent is pushed into the un-visited territory and increases the chance of finding the exit.

2.4 Exploring with Small Modules

The proposed method works best when we have $n \geq x$ and $m \geq y$ which means $\mathbf{M}$ is at least as big as $\mathbf{E}$ in both dimensions. When these conditions do not hold, we have to deal with the problem of small modules which is akin to have limited recollection of the past locations. To demonstrate this, Figure 3.5 provides a complete routing example with smaller modules in the presence of closed blocks. Tracking the conditions of $\mathbf{M}$ when $\mathbf{A}$ goes from b_{00} to b_{30} is similar to previous examples. In this state, $\mathbf{A}$ can go from b_{30} to either b_{40} or b_{20}. The b_{40} has not been visited before but cell c_{00} which is responsible for firing b_{40} has value 1 which means it has been visited. Therefore, $\mathbf{A}$ instead of giving the priority to b_{40}, gives the same priority to b_{20}, b_{40} and randomly chooses b_{20} instead.

As we can see, the increase of weights for c_{00} in b_{00} affected the weight of this cell for b_{40} which are two different places with different conditions. This problem may affect the completeness of the method. It also potentially compromises the second proposed method for dealing with closed paths. To solve the first problem, we bring a degree of uncertainty into the weights we have discussed so far. In this way, we make sure that when each place is not represented by a particular cell, we do not over-rely on the results of the weights.

Suppose $\mathbf{A}$ wants to move from $b^{(1)}$ and it has to choose $b^{(2)}$ between its neighbors. To choose the next destination, first we put the candidate neighbors in a set $\mathbf{S}$. The order of the elements in $\mathbf{S}$ have an important role in this process. We define function $\mathbf{S}.sort(c.w.\{p_1|p_2\})$ which sorts the elements of $\mathbf{S}$ based on the parameter $c.\{p_1|p_2\}$. In the first step, with $\mathbf{S}.sort(c.w.p_1)$, the elements are sorted in an ascending order based on p_1. In this way, we put the elements with lowest p_1 first which should have the highest chance to be selected as the next candidate.

In the next step, we use $\mathbf{S}.sort(c.w.p_2)$ to sort $\mathbf{S}$ again. This time, the cells which have similar p_1 values are sorted based on p_2 values in a descending order.

In this regard, the first element of $\mathbf{S}$ is the one that should have the highest priority, and the last element should have the lowest probability. If $\mathbf{S}$ has k elements, we define the probability vector $\mathbf{S}.P_i = [\mathbf{S}.P_1, \dots \mathbf{S}.P_k]$ which represents the probability of choosing each element. Consequently, for the probability vector $\mathbf{S}.P$, we devote the probabilities by Pareto Distribution [80] which assigns the highest probability to P_1 and lowest probability to P_k, as follows,

$$\rightarrow \mathbf{S}.P_i = \begin{cases} (\frac{1}{i})^{\alpha} & i > 1 \\ 1 & i \leq 1 \end{cases} \tag{2.5}$$

in which α is a positive parameter. When $i = 1$ the probability is one, and for $i > 1$ the probability is less than one but not zero.

To choose the $b^{(2)}$, we start from the last element of $\mathbf{S}$ and go to the first element of $\mathbf{S}$. Each of these elements has a chance to be selected based on their $\mathbf{S}.P_i$. In this way, based on parameter α, the last element have lowest chance to be taken, and the first element is selected with the probability of 1. It should be noted that because the $\mathbf{S}.P_i$s do not necessarily add up to 1, $\mathbf{S}.P$ is not a probability distribution. However, this does not affect the method, because all we need is a proper degree of uncertainty which is achieved with this method. Algorithm 2 provides an overall process for the navigation. As we can see, the initialization and usage of the $\mathbf{S}.P_i$ is necessary only for small modules.

Proposition 1 *The proposed method in Algorithm 1 is complete.*

Proof: In a two-dimensional region, suppose that we have the agent $\mathbf{A}$ at a particular location (x, y). We know that $\mathbf{A}$ moves randomly through the integer lattice $\mathbf{E}$ to any of the adjacent vertices which is specified by probability set $\mathbf{S}.P_i = [\mathbf{S}.P_1, \dots \mathbf{S}.P_k]$ at each time step τ. In the proposed method, we choose next block based on the set $\mathbf{S}$ in which the blocks with smallest $c.w$ are more likely to be chosen. Another possibility is when size of the $\mathbf{M}$ is very small (for instance it has only one neuron) or for other reasons like beginning of the search, $c.w$s have equal values in which we have random tie-breaking.

Considering this fact, we do not include some of the directions at any particular point (x, y)

Algorithm 1 Pseudocode for overall process of the navigational model.

```
// initialize module and get current state of the agent.
initialize(M)
current-state = get-state(A)
// (small modules only) provides randomness for selection.
$\mathbf{S}.P_i = [\mathbf{S}.P_1, \ldots \mathbf{S}.P_k]$
// while current state is not equal to the goal, continue.
while (current-state != end-state)
    // get neighbors of the current-state.
    neighbors = select-neighbors(current-state)
    // update $c.w.p_2$ for affected cells in module.
    update-views(M)
    // prioritize neighbors based on $c.w.p_1$.
    neighbors = prioritize 1(neighbors)
    // prioritize neighbors based on $c.w.p_2$.
    neighbors = prioritize 2(neighbors)
    // choose the neighbor with highest priority.
    current-state = choose-next-state(neighbors, $\mathbf{S}.P_i$)
    // update $c.w.p_1$ for affected cells in module.
    update-weights(M)
```

at time τ. However, in the worst case, in which **A** chooses a wrong path, the destination e is not reached, and other directions also did not give proper results, **A** meets point (x, y) again at time $\tau + \Delta$. In this step, the other directions which were not considered before, have a higher or equal probability of being chosen. The iteration of this process, selects all possible directions of (x, y), in different time intervals. Since the number of the blocks is limited and in the worst case, all of the adjacent vertices of each block are selected, the method eventually selects all of the possible open blocks which proves completeness of the method. ■

The last problem that remains is the ability to recognize the closed paths which is not solved by the above methods. To this end, we initialize $\mathbf{A}.counter$ with a ”time to live” (ttl) value. The initialization of $\mathbf{A}.counter$ increases the longevity of search time based on requirements of different scenarios. Note that limiting the search to ttl endangers the completeness of the method. As a result, the ttl value should be set cautiously. If the value is unnecessarily big, it may waste the agent's time when there is no possibility of reaching the destination. If the value is small, the agent may stop the search before trying all of the possible open blocks.

	$D(x \times y)$[a]	P_θ [b]	**Size** [c]	h	**Memory** [d]	ttl	$P.v$	α
				Scenario 1				
DFS	(10x10)	0	100	N/A	∞	N/A	N/A	N/A
BFS	(10x10)	0	100	N/A	∞	N/A	N/A	N/A
LRTA*	(10x10)	0	100	N/A	∞	N/A	N/A	N/A
Q-Learning	(10x10)	0	100	N/A	∞	N/A	N/A	N/A
GNM	(10x10)	0	100	6	100	∞	1	0
				Scenario 2				
DFS	(10x10)	20	100	N/A	∞	N/A	N/A	N/A
BFS	(10x10)	20	100	N/A	∞	N/A	N/A	N/A
LRTA*	(10x10)	20	100	N/A	∞	N/A	N/A	N/A
Q-Learning	(10x10)	20	100	N/A	∞	N/A	N/A	N/A
GNM	(10x10)	20	100	6	100	∞	1	0
				Scenario 3				
DFS	(50x50)	15	2500	N/A	∞	N/A	N/A	N/A
BFS	(50x50)	15	2500	N/A	∞	N/A	N/A	N/A
LRTA*	(50x50)	15	2500	N/A	∞	N/A	N/A	N/A
Q-Learning	(50x50)	15	2500	N/A	∞	N/A	N/A	N/A
GNM	(50x50)	15	2500	26	2500	∞	1	0
				Scenario 4				
DFS	(100x100)	15	10000	N/A	∞	N/A	N/A	N/A
BFS	(100x100)	15	10000	N/A	∞	N/A	N/A	N/A
LRTA*	(100x100)	15	10000	N/A	∞	N/A	N/A	N/A
Q-Learning	(100x100)	15	10000	N/A	∞	N/A	N/A	N/A
GNM	(100x100)	15	10000	52	10000	∞	1	0

a- Maximum Dimensions of x and y. c- Maximum number of blocks
b- Probability of closed blocks). d- Maximum allowed memory

Table 2.2: Four different scenarios for the experiments.

2.5 Results

In the simulations, we analyze the expeditions of $\mathbf{A}$ from block $b^{(1)}.status = $ *start* to block $b^{(2)}.status = $ *end*. The result section is designed to compare the memory usage and algorithmic steps of agents with comparable methods. These methods should assume unknown environments, unknown goal positions and physicality of agents. The ruling out of the incompatible methods (see Introduction), puts GNM next to four other algorithms. Two of these algorithms are prominent blind search methods from graph theory, DFS, and BFS with backtracking feature. We use the recursive feature of these algorithms to enable $\mathbf{A}$ to find its way to the open blocks through the blocks that it already has visited.

The BFS in this regard is similar to Iterative Deepening Depth-First Search (IDDFS) algorithm [58]. The DFS with backtracking feature is usually called Backtrack algorithm. We also compare the method with Uninformed LRTA* and Q-Learning which are suitable for our experiments. In LRTA*, we use 0 as a constant heuristic value. For Q-Learning, an efficient implementation is used where there are no R matrices (rewards are presented on-sight). The memory allocation of Q matrices also are not predetermined, and blocks take space when it is required (on-sight).

The main comparisons are based on the results of 1000 experiments with the same environments for all algorithms. We choose a high number of experiments (1000) to divulge the pros and cons of each method. In these experiments, we use the scenarios of Table 2.2 to analyze the algorithms. Each feature represents various states including different locations of start and end points. Unlike GNM where the maximum size of memory can be determined, IDDFS, Q-Learning, Backtrack and LRTA* algorithms should have uncapped memories. These scenarios are used for comparisons between these four algorithms with GNM. We will also use Table 2.2 to compare the results where the maximum memory is capped. More explanations about these scenarios with capped memories are given in Section 2.5.4.

The features of each scenario are selected for different objectives. The scenario 1 has the smallest environment without any closed blocks. The purpose of scenario 2 is to keep the scales of scenario 1 but to add closed blocks randomly to the environments. In scenario 3, we use larger environments which also have some probability of closed blocks. In the transition from scenario 1 to scenario 2, the probability of closed blocks have changed, but scales of the environments are equal. In the transition from scenario 3 to scenario 4, the probability of closed blocks remained intact, but the scales are changed. In scenario 4, we have environments with scales of 100×100 which are the biggest environments we use in these experiments. The probability of closed blocks is 15%, which is similar to scenario 3. Other parameters are the same. It should be noted that the different scales ($D(x \times y)$) of these environments is also of interest.

Next to the main experiments which have general settings, we also provide four more scenarios to represent specific environments. For the proof of concept, we have two scenarios which perform the real transformation between 3D environments into 2D topological maps with SLAM methods. The first scenario is a combination of the Gazebo SLAM simulator [81, 82], using Turtlebot 2.0 with the proposed method (SLAM-Sim). The second scenario is a combination of the real world Turtlebot 2.0, with the proposed method in our office room, where we provided enough roaming space (SLAM-Real). The main results for these scenarios are for five experiments.

The other two scenarios are for environments that have specific shapes. The shapes that we use are circular and logarithmic spiral. These two environments can provide an overall representation for other shapes (e.g., Triangular, Oval, Hexagonal) as well. The circular mazes feature the shapes that have narrower roaming areas compared to the logarithmic spiral, which has wider roaming areas. The main results for these scenarios are for 1000 experiments. In the following, for all of the experiments, there is at least one path from start to the end blocks. As for the α value, it only needs to be set when the memory is capped. Another note is for the value of $P.v$ which as we mentioned can take any integer value starting with 1. However, for a fair comparison with other methods, only the value 1 is used throughout the experiments. For the SLAM scenarios, the viewing distance of the SLAM determines the value of $\mathbf{A}.v$.

2.5.1 Environmental Setup

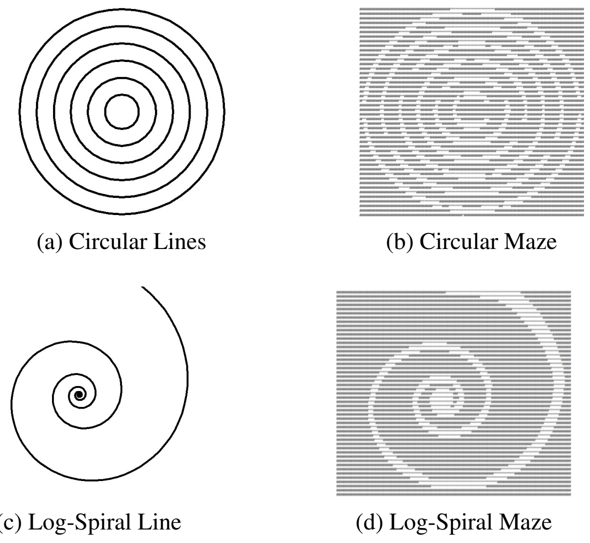
To implement the experimental environment for navigational algorithms (navigational software), we use Java. The infrastructure of the program consists of five main components, the main (Main.java) and commands (SimCommands

.java) classes which control the overall flow of the program. The maze class (Maze.java) creates the maze rooms. The Block class (Block.java) which creates blocks used by the maze and the GridCell class (GridCell.java) which creates blocks for the grid cells. The Maze class is created with flexibility in mind. The constructor of the class takes four arguments. The first two arguments for determining dimensions of the maze, the third to determine the probability of each block to be closed (for constructing the walls) and the fourth to set a seed value for random numbers (required for our simulations). As a result, in the main class, we can specify the dimensions of the maze, the probability of closed blocks and provide exact environments for all algorithms. For the navigation itself, we have created multiple classes which are in direct connection to the infrastructure classes. These classes implement the navigational algorithms to lead the agent from a start position to an end position.

The Block class has important information for each block. These values include dimensions of each block, required values for information about the agent in each block, and the status of the block. The status value is useful to standardize the input of the program. In our program, we demonstrate each block as:

- Status 0: which corresponds with character '#' and means those blocks which are open.
- Status 1: which corresponds with character '*' and means those blocks which are closed.
- Status 2: which corresponds with character 's' and means those blocks which are starting points.
- Status 3: which corresponds with character 'e' and means those blocks which are ending points.
- Status 4: which corresponds with character '.' and means those blocks which have been visited in the search.
- Status 5: which corresponds with character '-' and means those blocks which have been viewed before.

We can use this infrastructure to create the experiments for defined scenarios in Table 2. For other scenarios, the ability to read from the text files is added to the program. In this regard,



(a) Circular Lines (b) Circular Maze

(c) Log-Spiral Line (d) Log-Spiral Maze

Figure 2.5: A transformation of images, created with Turtle package in python (a,c) into our input text file for navigational software (b,d). The image shows that the blocks of the input text file are randomly opened (change '*' chars to '#' chars) to connect all of the open blocks to each other.

the environments from images can be converted and imported into the program. We use this mechanism to create and import mazes with specific shapes. For this purpose, the output of the maze generators is converted to the standard assumptions of the block status explained above. For instance, a closed block in the maze should be presented by '*' char and the open blocks should be presented by '#' char.

In Figure 2.5, we see the process of creating input data for our program. In this process, first, we created Circular and Log-Spiral lines using Turtle package in python [83]. In the next step, we converted the images to text formats with an image to text art converter script. In our script, a list of chars represent greyscale values from darkest to brightest. For conversion, the image will be divided into rectangles. In the end, by setting the right colors of the rectangles to the correct text values (for instance, setting black rectangles to '*' and white rectangles to '#') we can create an input data which is readable by our program. In the Circular maze, we randomly changed some '*' blocks to '#' blocks. In this way, we could connect all of the open blocks to each other. Also, for Log-Spiral, by manually changing some '#' blocks to '*', we made the lines thicker towards the tail of the Log-Spiral line.

For the proof of concept, we also combine SLAM with the proposed method. In this regard,

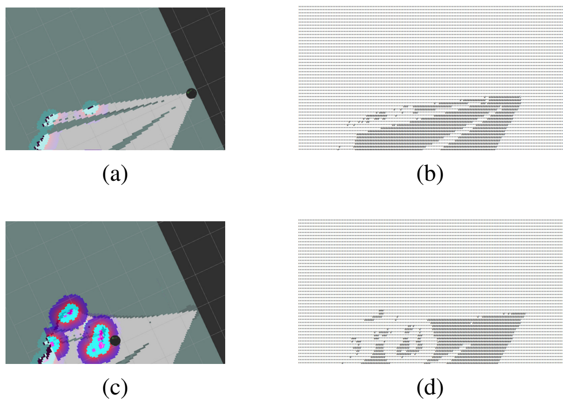


Figure 2.6: As the robot moves between two locations, with SLAM, it can update its view of the Real-World (a,c). Both views can be converted into our input text file for navigational software (b,d). The image shows the update of the map from moving the SLAM robot.

we use a very basic approach which is compatible with our navigational software. In practice, more advanced methods such as [84, 85, 86] should be used to provide a mapping (lattice) representation of the real environment. The scenarios 5 and 6 are done with the help of TurtleBot 2.0 toolsets[3]. The same tools are used for connecting the simulation environments and the real world TurtleBot 2.0. This is done by ROS toolkit for Ubuntu operating system. With this setup, we can get the SLAM map from the TurtleBot and also guide the robot into our desired directions with ’u’, ’i’, ’o’, ’j’, ’k’, ’l’, ’m’, ’,’, ’.’, keyboard maps. The process is the same for both 3D simulator and the real world TurtleBot.

For navigation, we use four steps, 1- we receive the information of the environment from TurtleBot in the form of an image. 2- we use the image-to-text art converter script, to feed the image as an input to our navigational software. 3- we take several steps with the navigational software. 4 - we move the robot remotely with keyboard maps to the current destination that navigational software is pointing. To perform the experiments, we continuously update the map of navigational software from TurtleBot and receive the next step from the navigational software to move the TurtleBot until we find the destination. A sample of this conversion is shown in Figure 2.6. As we can see, SLAM updates its view of the environment when the robot moves through a certain path. It also shows the respective conversions of the SLAM environment to the input file of the navigational software. The same process is performed for

[3]TurtleBot and Turtle package in python are two different toolsets and are not related to each otehr.

the Gazebo SLAM simulator. All of these conversions and respective movements are done with the supervision of a person and are compatible with the design of our navigational software. As long as the environments are kept simple, such conversions are feasible. In more complex environments, better solutions should be used.

2.5.2 Test of Speed

For analyzing test of speed, each step allocates one time unit. The number of steps that the agent has to take from start state to the end state determines the speed of the agent. Table 3.1 provides the actual speed performance of our method and the other algorithms considering scenarios 1-4. In the first scenario, for mean values, GNM is at least 2.67 times better than LRTA*. This is followed by $5.89, 7.78$ and 44.67 times better speed than Backtrack, Q-Learning, and IDDFS. The same pattern continues for standard deviation where GNM is at least 3.16 times less than LRTA* and at most 44.67 times less than IDDFS. For the maximum value, the LRTA* provided a higher number than Backtrack. The GNM is at least 3.07 better than Backtrack and at most 58.43 times better than IDDFS.

In the second scenario, for mean values, GNM is at least 3.05 times better than LRTA*. This is followed by 4.74, 8.45, and 30.57 folds better speed than Backtrack, Q-Learning, and IDDFS. For the standard deviation, GNM is at least 3.79 times less than Backtrack, and at most, 40.08 times less than IDDFS. For the maximum value, the LRTA* again has a higher value than Backtrack. The GNM is at least 2.94 better than Backtrack and at most 74.48 times better than IDDFS.

In scenario 3, the dimensions of the environments are much bigger than previous scenarios from 10×10 up to 50×50. The probability of each block to be closed is 15%, which is comparatively less than the probability of closed blocks in scenario 2. With the expansion of the environments, the IDDFS algorithm performed worse and became much slower compared to its results from previous scenarios. In this scenario, for mean values, GNM is in order, 3.65, 5.73, 9.00, and 139.05 times better than LRTA*, Backtrack, Q-Learning, and IDDFS. For standard deviation, GNM has at least 4.20 times less value than Backtrack and at most, 367.54 times less value than IDDFS. Respectively, for maximum value, GNM is at least 3.11 times better than Backtrack and most 560.21 better than IDDFS.

	Network	Scenario1	Scenario2	Scenario3	Scenraio4
Mean	GNM	28	35	746	4981
	LRTA*	75	107	2723	19196
	Backtrack	165	166	4275	18066
	QLearning	218	296	6715	55416
	IDDFS	1251	1070	103738	8282716
STD	GNM	25	34	893	3774
	LRTA*	79	202	4936	25235
	Backtrack	103	129	3756	12123
	QLearning	257	422	11544	52781
	IDDFS	1559	1363	328214	12430392
Max	GNM	143	174	4725	26318
	LRTA*	973	3580	44995	251570
	Backtrack	440	513	14699	58029
	QLearning	2228	3803	129144	448082
	IDDFS	8356	12960	2647013	64535862
Mode	GNM	2/147	2/127	2/36	1720/3
	LRTA*	2/52	2/61	4/25	6112/2
	Backtrack	2/26	7/40	7/32	14086/2
	QLearning	2/27	2/28	2/23	16440/2
	IDDFS	27/84	27/55	27/120	1126435/2

Table 2.3: Mean, standard deviation (STD), max and mode results for the test of algorithmic steps on scenarios 1,2,3,4.

In scenario 4, the IDDFS algorithm continued the trend, and with the bigger environments became much slower compared to other algorithms. An interesting observation is that Backtrack for the first time shows better results than LRTA*. The Q-Learning provided the worst results which show its slow learning process in bigger environments. In scenario 4, for mean values, GNM is in order, 9.55, 2.20, 17.02, and 2452.15 times better than Backtrack, LRTA*, Q-Learning, and IDDFS. For standard deviation, GNM is in order 3.21, 6.68, 13.98, and 3293.69 times less than Backtrack, LRTA*, Q-Learning, and IDDFS. Respectively, for the maximum value, GNM has in order 9.55, 2.20, 17.02, and 2452.15 times better values compared to the Backtrack, LRTA*, Q-Learning, and IDDFS.

For the mode, each cell has two values. The left side represents the mode value, and the right side represents the number mode repetition in 1000 trials. As we can see, with bigger scenarios, the number of repeats for each mode value is decreased dramatically. This is more or less accurate for all algorithms. The GNM, started, with a mode of 2 steps for scenario 1, which is repeated 147 times. For scenario 4, the mode is 1720 with only 3 repetitions. LRTA* started with a mode of 2 steps for scenario 1 with repeat value of 52 and ended with a mode of

	Network	Circular	Log-Spiral	SLAM-Sim	SLAM-Real
Mean	GNM	5140	7801	117	25
	LRTA*	69130	191613	735	45
	Backtrack	14125	17982	1052	182
	QLearning	159485	328399	1044	207
	IDDFS	2826254	13774969	9201	874
STD	GNM	4870	8272	85	10
	LRTA*	100981	475425	417	50
	Backtrack	9497	13942	750	117
	QLearning	190104	649376	425	121
	IDDFS	3406973	18754080	7107	908
Max	GNM	27614	53517	263	40
	LRTA*	1009049	4669480	1337	142
	Backtrack	37928	50749	1947	299
	QLearning	1464748	5033045	1624	340
	IDDFS	19867679	100916108	22759	2527
Mode	GNM	23532/12	899/6	N/A	24/2
	LRTA*	12026/12	220/6	N/A	16/2
	Backtrack	1820/12	36681/7	N/A	39/2
	QLearning	104830/12	2451/6	N/A	340/2
	IDDFS	874786/12	183323/6	N/A	881/2

Table 2.4: Mean, standard deviation (STD), max and mode results of extra scenarios for the test of algorithmic steps.

6112 steps for scenario 4 with a repeat value of 2. Backtrack started with a mode of 2 steps for scenario 1 with repeat value of 26 and ended with a mode of 14086 steps for scenario 4 with repeat value of 2. The IDDFS started with a mode value of 27 and a repeat value of 84. It ended with a mode value of 1126435 and a repeat value of 2.

We further analyze the algorithms with other four environments in Table 2.4. For the circular environment (Figure 2.5.b) the mean value is up to 3.65 fold better than IDDFS and down to 2.74 fold better than Backtrack. For standard deviation, GNM is up to 699.58 times less than IDDFS and down 1.95 times less than Backtrack. The maximum value of GNM is up to 719.47 fold better than IDDFS and down to 1.37 fold better than Backtrack. For the Log-Spiral environment (Figure 2.5.d), the mean value is up to 1765.79 fold better than IDDFS and down to 2.30 fold better than Backtrack. For standard deviation, GNM is up to 2267.17 times less than IDDFS and down 1.68 times less than Backtrack. Respectively, for the maximum value, GNM is up to 1885.68 times better than IDDFS and down 0.94 times better than Backtrack.

For the test of speed in SLAM environments, we use the number of steps in navigational software. In this way, we provide an accurate number for the algorithmic steps and also prevent human intervention for the calculations. For the SLAM-Sim, for the mean values, the GNM is in order, 6.28, 8.99, 8.92, and 78.64 folds better than LRTA*, Backtrack, Q-Learning, and IDDFS. For the standard deviation, the GNM is at least 4.90 times less than LRTA* and at most 83.61 times less than IDDFS. For the maximum values, the GNM is in order, 5.08, 7.40, 6.17, and 86.53 folds better than LRTA*, Backtrack, Q-Learning, and IDDFS. There was no mode for the SLAM-Sim, which is normal because we only had 5 number of experiments. For SLAM-Real, the mean value is at least 1.8 fold better than LRTA* and at most 34.96 fold better than IDDFS. For the standard deviation, GNM is at least 5 fold smaller than LRTA* and at most 90.8 fold smaller than IDDFS. For the maximum values, GNM is at least 3.55 fold better than LRTA* and at most 63.17 fold better than IDDFS. Here the algorithms have mode values which is expected because the environment was relatively smaller than the SLAM-Sim.

Overall the results here more or less confirm our previous results. Similar to Table 3.1, IDDFS is markedly slower than other algorithms. In the bigger environments, the difference from our algorithm with others become more apparent. As we have seen, with the less roaming environment, the results of the Backtrack algorithm improves considerably and lessens its distance to the GNM. The reason is that Backtrack is biased toward narrower environments but does not perform as well in more general settings.

2.5.3 Memory Usage

For the memory comparisons, the first step is fair assumptions of memory allocations for all algorithms. To this end, efficient implementations of the algorithms are required. In GNM, we consider that whenever a cell has been fired, one memory unit should be dedicated to that cell. As described before, the firing occurs for two purposes, 1) dispatch to another place 2) viewing another place. To simplify the process, we consider the memory that is used for either or both of the two options to have the same cost. This premise is an acceptable assumption because we need only one variable for implementing them. Comparatively, all of the variables of LRTA*, Backtrack, Q-Learning and IDDFS algorithms for marking and storing the places or policies are on-sight, which helped us to preserve compatibility and fairness between different algorithms.

In this regard, the memory usage of GNM is counted for the expansion of **M**. However, the memory usage of LRTA*, Backtrack, and IDDFS algorithms are for the expansion of **E**.

	Network	Scenario1	Scenario2	Scenario3	Scenraio4
Mean	GNM	45	37	754	4610
	LRTA*	48	37	749	4593
	Backtrack	103	68	1673	7942
	QLearning	73	61	1297	10144
	IDDFS	37	30	250	4182
STD	GNM	30	24	712	2211
	LRTA*	30	24	693	2306
	Backtrack	57	42	1254	4342
	QLearning	49	42	1429	4479
	IDDFS	31	24	516	2582
Max	GNM	100	81	2129	8505
	LRTA*	99	79	2123	8512
	Backtrack	198	145	3749	15139
	QLearning	163	147	4404	17954
	IDDFS	117	123	2358	10525
Mode	GNM	6/124	6/64	15/25	2126/3
	LRTA*	2/52	2/61	4/25	6769/3
	Backtrack	120/33	8/42	8/34	13014/3
	QLearning	2/28	2/29	2/23	11852/2
	IDDFS	9/147	8/91	9/109	2136/3

Table 2.5: Mean, standard deviation (STD), max and mode results for for the test of memory on scenarios 1,2,3,4.

Backtrack and IDDFS also need the memory required to preserve the longest path for the recursiveness. For Q-Learning, the unit cost is for blocks of Q matrix, which as mentioned, reserves memory when required. For analysis of the allocated memory, observe Table 2.5. The structure is similar to speed analysis. For scenarios 1-4, in each 1000 experiments, a snapshot is presented which provides the overall results to that point. As it is exhibited, in scenario 1, the Backtrack algorithm always consumed the highest memory for mean, standard deviation, and maximum values, which is followed by Q-Learning. Among other algorithms, IDDFS has the smallest footage of using memory. This outcome is followed by GNM, which takes the second place and closely followed by LRTA*.

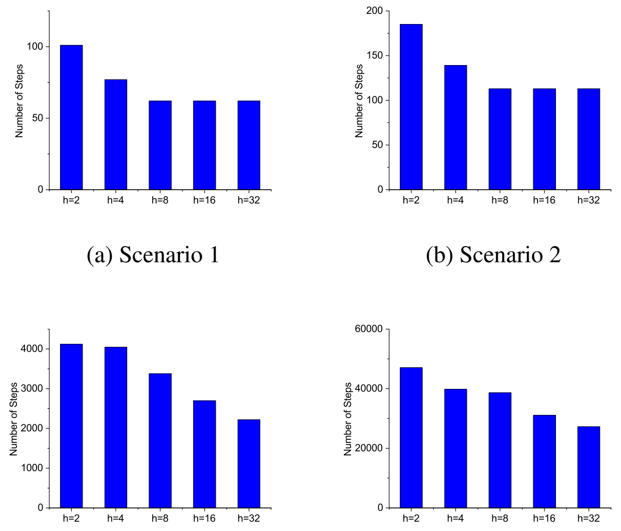
In scenario 2, the memory usage of IDDFS is similar to scenario 1 and less than other algorithms. The GNM and LRTA* in this scenario have close results. The Backtrack is in the last place. However, it could slightly compensate and reduce its distance from other algorithms, while Q-Learning became worse. For the maximum value, Backtrack is slightly better than Q-Learning. In scenario 3, with the increase of scales of the environments, it should not be surprising that memory is also increased for all algorithms. However, the increase of memory

	Network	Circular	Log-Spiral	SLAM-Sim	SLAM-Real
Mean	GNM	2893	3936	151	36
	LRTA*	2965	3931	241	29
	Backtrack	5246	6630	374	74
	QLearning	8732	12072	368	71
	IDDFS	1854	2885	140	26
STD	GNM	1625	2283	84	17
	LRTA*	1571	2277	53	22
	Backtrack	2852	3874	194	46
	QLearning	4910	7352	124	25
	IDDFS	1258	2186	113	15
Max	GNM	5741	7835	277	66
	LRTA*	5737	7835	308	69
	Backtrack	10457	13334	561	133
	QLearning	17257	25275	520	87
	IDDFS	5214	9229	359	49
Mode	GNM	5649/12	2535/7	N/A	30/2
	LRTA*	1429/12	2090/8	N/A	16/2
	Backtrack	1153/12	5944/6	N/A	20/2
	QLearning	11972/12	1267/6	N/A	84/2
	IDDFS	347/13	763/7	N/A	30/2

Table 2.6: Mean, standard deviation (STD), max and mode results of extra scenarios for the test of memory.

for IDDFS is less than others. Similar to scenario 2, the results between LRTA* and GNM is close where LRTA* slightly edges GNM. The distance between Backtrack, Q-Learning, and other algorithms is increased while Q-Learning performed better than scenario 2. In scenario 4, IDDFS unsurprisingly is more efficient. For the LRTA* and GNM, this scenario is similar to scenario 3. The surprising element is Q-Learning performing worse than Backtrack, which supports the results of speed analysis for scenario 4. This outcome is due to the severity of the slow learning process of Q-Learning in bigger environments.

The results of the other four scenarios is presented in Table 2.6. As it is shown, for the Circular and Log-Spiral scenarios, the Q-Learning algorithm almost always consumed the highest memory, which is followed by Backtrack. For GNM and LRTA*, in Circular environment, GNM performed slightly better, and for Log-Spiral, LRTA* showed slightly better results. In the SLAM-Sim and SLAM-Real environments, Q-Learning and Backtrack performed almost similarly. Respectively, GNM and LRTA* also provided close results. Overall, by considering all of the scenarios, IDDFS has the smallest footage of using memory. This outcome is followed by GNM and LRTA* which are almost similar and take the second place. Q-Learning and Backtrack also show similarities, but in general, Backtrack consumed less memory. Note



(a) Scenario 1 (b) Scenario 2

(c) Scenario 3 (d) Scenario 4

Figure 2.7: Scenarios 5,6,7,8 - dynamic memory of GNM.

that in these experiments, SLAM-Sim had no mode values. This is expected because we only performed five experiments. For SLAM-Real, because the size of the environments was relatively smaller, even with five number of experiments we had mode values.

2.5.4 Scenarios 5,6,7,8 - Dynamic Memory

We know that the dynamic nature of GNM, allows us to cap the maximum number of available memory for the search. Previously, we compared GNM with available search algorithms where each place had a specific cell which would fire only for that place. In this section, we compared GNM for scenarios where the maximum size of the modules is limited to an upper bound. These results are noteworthy because while LRTA* and GNM, were competitive on memory usage, the IDDFS took the lead. The IDDFS however, is impractically slow. In these experiments, we focus on the dynamic nature of the memory in GNM and show that even on memory usage, GNM can provide a better solution.

For these experiments, we use the same scenarios which are defined in Table 2.2. The focus of these experiments is the value of h, which is used to cap the maximum available memory. The method described in Section 4.2.3 is also added to provide a degree of uncertainty for decisions. Figure 2.7 illustrates the effects of memory caps for different environments based on different h values. As we can see, the value of h, which determines the maximum available memory is increased exponentially. The GNM, consumes the available memory until there is no need for more memory space.

In our experiments, even with the lowest memory cap, the speed is 12.38 fold (scenario 1) to more than 175.80 fold (scenario 2) better than IDDFS in mean values. This apparent gap shows that even with extreme cases of lack of the memory, GNM beats its closest rival. To analyze the persistence of the proposed method, we compare the increase of the speed to the increase of the memory. In scenario 1, with $h = 8$, the algorithm converges to its highest required memory. In this case, the memory usage is an increase from 12 $(h = 2)$ to 240 $(h = 8)$. The memory has increased 20 times, however, the maximum speed which could be gained was only 1.62 times better. For the second scenario, the increase of the memory is the same, and the maximum speed which could be gained was only 1.63 times better.

The similar pattern is repeated more or less for other scenarios as well. For the third scenario, the memory is increased from 12 $(h = 2)$ to 2256 $(h = 24)$ which is an increase of 188 times more memory space. Respectively, the increase of the space is only 1.85 times. For the fourth scenario, the memory is increased from 12 $(h = 2)$ to 4032 $(h = 32)$ which is an increase of 336 times more memory space. Respectively, the increase of the space is only 1.72 times. These outcomes show the persistence of the GNM, which provides a reasonable speed despite the lack of memory. While these results are encouraging, to achieve the highest speed, the availability of the memory is still necessary.

Chapter 3

Automated Supervised Learning

3.1 Greedy AutoAugment

Data augmentation is an important technique that can help to improve the performance of various data analysis algorithms in the presence of insufficient data. For instance, a common practice in medical applications is class-specific data augmentation. In this case, gathering sufficient labeled data to train a deep neural network model is impractical [87, 88]. This is a classic type of long-tail distribution data, which is also prevalent in natural images [89, 90] and can be addressed by data augmentation methods [91]. Other important usages of data augmentation methods include unsupervised learning [92, 93, 94, 95] and in the improved training of generative adversarial networks [96, 97, 98, 99]. In this section, we focus on the problem of data augmentation for image classification . The main goal is to increase the accuracy of image classification by applying the right augmentation techniques on training data.

Data augmentation in image classification is directly related to image-based object transformations. In the classification process, it is desirable to take into account a variety of such transformations to improve image classification. In other words, we want the perception of an object to be invariant to the properties such as scale, brightness, rotation, and viewing angle. Estimating important object transformations and applying them in the learning process is a critical problem in Artificial Neural Networks (ANNs). For instance, it is desirable that a network, after learning an object from its original form, recognizes the same object in a modified location where its scale, rotation, and other properties have been changed. Currently, there are two ways to deal with this problem. First, by designing network architectures that can inherently be invariant to important image-based object transformations. Second, with the use of data augmentation methods.

The most basic network which considers the transformations of the input data is the Convolutional Neural Network (CNN). The CNN architecture, with the concept of convolutional layers, tries to be translation invariant [100, 101]. This network was very successful in its approach and has been used as a basis for the development of more advanced architectures [102, 103, 104]. Another example of this approach is CapsuleNet, which tries to find the relevant pose information automatically [105, 106, 107]. While the design of CapsuleNet improved the results of basic datasets [108], unfortunately, it could not improve the accuracy for more complicated datasets such as ImageNet [109].

The second method for considering different transformations of the input data is to use data augmentation. In this method, the objective is to achieve invariance by applying different image transformations such as geometry transformation, kernel filtering, color transformation, image mixing, random erasing [110], etc. The main advantage of this method is simplicity and supporting all forms of ANN architectures. Additionally, there is a possibility to use transformation techniques in which current ANN architectures do not support.

One of the most important factors for data augmentation techniques is the constraint on the number of possibilities for applying augmentation techniques. In this regard, only a subset of the possible techniques can be used for data augmentation. Therefore, a search mechanism is needed to find the best possible techniques. The most common method to find the best data augmentation techniques is to find them manually [101, 111, 112] which needs prior knowledge and expertise. Recently, the AutoAugment [113] is proposed to automate the process of finding the best augmentations. In this method, finding the augmentation policy is reduced to a discrete search problem over various augmentation techniques, each having hyperparameters of the probability of applying the operation and the magnitude to which the operation is applied. Because of the computational requirements for searching with AutoAugment, this method relies on transferability of the augmentation techniques. This means that the policies that are found with one dataset and architecture can be used for similar datasets with different architectures [113].

Using the same policies which are found for a specific scenario and trying to use them for others is risky and may lead to worse results (see the Results). To solve this problem, it is desired to perform the search more effectively, so that searching can be performed for each dataset separately. In this section, we address the problem of how to effectively search for the most appropriate data augmentation techniques and apply them to training data. For this purpose, we propose Greedy AutoAugment. In this method, we develop a greedy-based search algorithm, which reduces the searching space from the exponential growth of possible trials to linear growth. With the proposed method, instead of searching for all possible outcomes, the search expands dynamically, and for each sub-policy, it is pushed towards trials with the best outcomes. To achieve this goal, we search among sub-policies with only two elements, the techniques of operations and the magnitudes for those techniques. The probability is applied after the search is finished with a methodology that gives weights to better policies. Our experimental results show that this approach is effective in providing higher accuracies. The Greedy AutoAugment, on four datasets, Tiny ImageNet, CIFAR-10, CIFAR-100, and SVHN, could reliably provide higher accuracies while using 360 times fewer computational resources.

3.1.1 Data Augmentation

Data augmentation refers to the practice of applying a series of standard transformations [114, 110, 115] to the given image data. In order to use these transformations, for each epoch in the training phase, a percentage of the data receives one or a combination of these techniques.

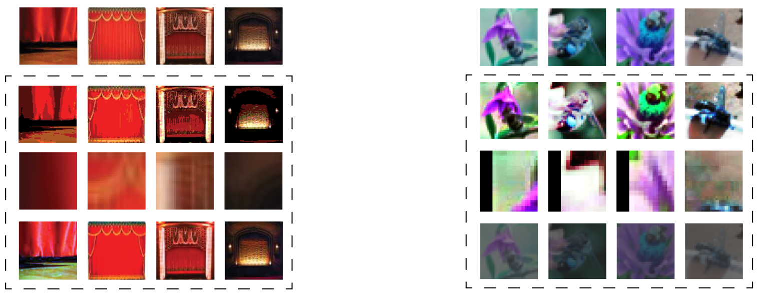

(a) ImageNet (b) CIFAR-100

Figure 3.1: Random data augmentation techniques applied to samples from two real datasets. Each row receives the same augmentation technique with different magnitudes.

The effect of applying data augmentations to the images from ImageNet and CIFAR-100 datasets are shown in Figure 3.1. In each row, one specific combination of augmentation techniques with different magnitudes is chosen randomly and is applied to the columns of images. As we can see, in some instances, the change is not noticeable, and in others, the change could completely change the original data. The goal is to discard combinations that would decrease the generalization ability of the network and select the best combinations that would increase it.

The augmentation techniques in their simplest form have been used in prominent artificial neural networks [101, 111, 112]. These networks mostly used a trial and error approach to manually find the best combinations. A more advanced approach is proposed in [113], which automates the searching process for finding the best augmentation techniques. There are also methods that do not perform a real search to find the best augmentations. Instead, these methods are created to be resilient against randomly selected augmentations [116, 117].

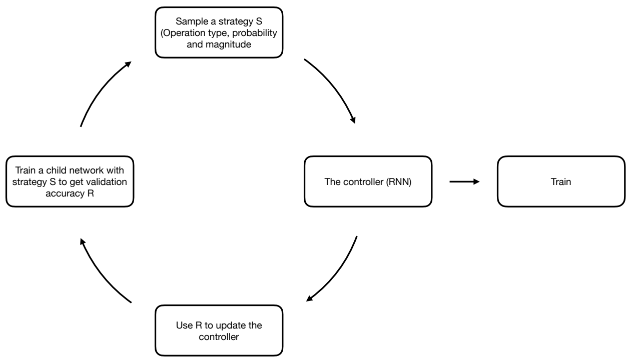

Sample a strategy S (Operation type, probability and magnitude
The controller (RNN)
Train
Use R to update the controller
Train a child network with strategy S to get validation accuracy R

Figure 3.2: The general scheme of AutoAugment algorithm.

Table 3.1: Augmentation techniques with their descriptions which are used in GAutoAugment.

	Technique	Description		Technique	Description
1.	FlipLR	Filliping the image along the vertical axis.	11.	Contrast	Changing the contrast of the image.
2.	FlipUD	Filliping the image along the horizontal axis.	12.	Brightness	Adjusting the brightness of the image.
3.	AutoContrast	Increasing the contrast of the image.	13.	Sharpness	Adjusting the sharpness of the image.
4.	Equalize	Equalizing the histogram of the image.	14.	ShearX	Sheering the image in horizontal axis.
5.	Invert	Inverting the color of the pixels in the image.	15.	ShearY	Sheering the image in vertical axis.
6.	Rotate	Rotating the image by certain degrees.	16.	TranslateX	Translating the image in horizontal axis.
7.	Posterize	Redicing the number of Bits for each pixel.	17.	TranslateY	Translating the image in vertical axis.
8.	CropBilinear	Croping with bilinear interpolation strategy.	18.	Cutout	Changing a random square patch of the image to gray pixels.
9.	Solarize	Inverting the color of all the pixels above a certain threshold.	19.	Blur	Blurring the image.
10.	Color	Changing the color balance of the image.	20.	Smooth	Smoothing the image(Low-pass filtering).

The searching process of the AutoAugment method heavily relies on NasNet [118] as a controller to direct the search. The controller predicts a decision by using a one-layer LSTM, which contains 100 hidden units and 30 units softmax predictions. The prediction is then fed into the next step as an embedding. In the end, the controller uses 30 softmax predictions in order to select the best policies. The LSTM should be in direct contact with some child networks to train its own network. The child networks are usually a subset of the training networks. The accuracy updates from child networks are used to update the main LSTM network. To use the accuracies, a policy gradient method called Proximal Policy Optimization algorithm (PPO) is employed. The whole process is shown in Figure 3.2. The strategies (policies) have three elements 1- data augmentation techniques, 2- the probability of applying each operation, and 3- the magnitude of the operation. The policy gradient receives the values of reward R to update the controller. The LSTM uses the softmax predictions of its trained networks to select the best polices. In the end, the best policies are used to train the actual network.

3.1.2 Searching Environment

To perform data augmentation on image data, we use policies. Let us define policy function $\hat{f}_i$ that may include one or more sub-policy functions. The sub-policy is used as the functionality of applying augmentation techniques on image data. Each sub-policy has three essential elements, 1- the augmentation technique, 2- the magnitude of the operation, and 3- the probability of applying the operation. For a complete list of augmentation techniques that we use in this section, see Table 3.1. The magnitude is the degree in which an operation is applied. For instance, in the rotate augmentation, the magnitude specifies how much we should rotate an image. The third element specifies the probability of applying the augmentation on the image.

We define each image as a multivariate data point **x**. For augmentation on image data we use $\hat{f}(\mathbf{x}) \rightarrow \mathbf{x}'$ which transforms the original image **x** into the augmented image **x**′. The $\hat{f}_i$ is separated into one or a multiple number of sub-policies. The sub-policy function is denoted by f. Each sub-policy receives three input values, $f(t, p, m)$, where the t, p, and m variables represent the augmentation technique, probability, and magnitude. The search space is defined as all of the possible combinations of concatenated sub-policies. The search space for a single f, includes all of the possible combinations of discrete values of t, p and m. Accordingly, the search space for any f is $(t_n \times p_n \times m_n)$ where t_n, p_n, and m_n are the maximum values for t, p, and m.

To represent the search space, we can divide it into different layers. A search space that has only one layer includes an augmentation function $\hat{f}_i$ that has only one f as its sub-policy. We know that the search space for sub-policy f has a maximum number of possible elements of $(t_n \times p_n \times m_n)$. The search space with two layers is defined for $\hat{f}_i$s that can concatenate two fs as their sub-policies. Respectively, the maximum number of possible elements for the search space expands to $(t_n \times p_n \times m_n)^2$. We generalize the number of layers for any search space with the variable ℓ. The ℓ is a discreet integer value such that $\ell \geq 1$. The search space is defined as concatenated layers, with a maximum size of $(t_n \times p_n \times m_n)^\ell$, where ℓ indicates the number of allowed sub-policies for $\hat{f}_i$.

The important point in the defined searching environment is that the search space expands exponentially. Particularly, given base $(t_n \times p_n \times m_n)$ the possible number of trials increases exponentially with the increase of ℓ. To solve this problem, we transform the base of the search space into a two-variable setup. Among the three variables, t, p, and m in the sub-policies, we do not search for the variable p and fixate its value to one. Instead of searching for the best probability values among sub-policies, we apply the probability with a methodology that gives more weights to the policies with better accuracy results. This simple design decision helps us to reduce the computational requirements of the searching algorithm considerably.

Even with reducing the number of variables for the base, the growth of the number of possible trials is still exponential. To tackle this problem, when going from one layer to the other (concatenating two sub-policies), we use a greedy search. To use the greedy search, we replace the reinforcement learning in AutoAugment with Breadth-First Search, which is an explorer for tree-based data structures. Accordingly, the Greedy Breadth-First Search is used to explore the defined layered environment. The new number of the possibilities is defined as follows,

$$\sum_{1}^{k} (t_n \times m_n) \tag{3.1}$$

In this notation, k is an arbitrary integer number, which indicates the number of iterations the algorithm is allowed to perform in the search. The higher values of the k, allow algorithm to perform more trials, which may lead to higher accuracy. Therefore, the value of k should be set based on the available computational resources. The greedy approach helps us to convert the exponential growth of base $(t_n \times p_n \times m_n)$ with ℓ to linear growth of base $(t_n \times m_n)$ times the value of k which allows $\hat{f}_i$ to potentially encompass many layers. For numerical values, we follow the discretization setup, which is introduced in AutoAugment. The number of augmentation techniques that we use in this section is $t_n = 20$. The discretization of probabilities is with $p_n = 11$ values with uniform space, and the discretization of magnitudes is with $m_n = 10$ values with uniform space.

3.1.3 Greedy AutoAugment

Training with the Greedy AutoAugment algorithm includes two main steps. The first step is to search and find the best $\hat{f}_i$s. The second step is to apply the $\hat{f}_i$s to $\mathbf{X}$ and increase the generalization ability of the network. To perform the greedy search, we use Algorithm 2. The input of the searching process is $\mathbf{X}$, and the output is $(\hat{f}_1, \ldots, \hat{f}_d)$, which are the best augmentation policies found in the search. The number of iterations of the algorithm needs to be at least equal to one ($k \geq 1$).

The lines 1-10 are for $k = 1$. The Breadth-First Search takes place in lines 1,2, where we go through all of the augmentation techniques and their respective magnitudes. In line 3, we use fixed value of 1 for the probability. In line 4, $\mathbf{X}$ is divided into two parts $\mathbf{X}_{tr}, \mathbf{X}_{te}$ for training and testing. In line 5, the sub-policy is considered as a complete policy. In line 6, sub-policy f(t,p,m) are applied on $\mathbf{X}_{tr}$. The classification of $\mathbf{X}_{te}$ is determined in line 7 and the score of the sub-policy and its respective score is stored. Line 9 is for the greedy part of the algorithm, where the policy with the highest score is stored to act as a base sub-policy for more iterations.

For the next step (lines 11-23) the algorithm is for iterations with $k > 1$. As it is shown in line 11, the algorithm repeats its steps until the counter is equal to k. In line 12, the $\hat{f}$ is equal to the best policy with the highest score, which is not selected before. In this way, the next iteration starts from a base that has the best score. As we can see in line 17, the f(t,p,m) is added on top of the best policy. The other parts of the algorithm are the same as the first part. In the end, we select the d best $\hat{f}_i$s, which shows the highest accuracy results among all policies. In Algorithm 2, the probabilities for all policies were always set to the value of one. To determine the p values for $\hat{f}_i$s we propose the following process.

The goal is that the policy which has a higher score to receive a higher probability. For this purpose, let us define $S_1, \ldots, S_{d/\lambda}$, where S_i, is a set with λ number of probabilities, $S_1 = \{p_1, \ldots, p_\lambda\}, \ldots, S_{d/\lambda} = \{p_{d-\lambda}, \ldots, p_d\}$. In this regard, we define vector $\vec{v} = [v_1, \ldots, v_{d/\lambda}, v_{(d/\lambda)+1}]$

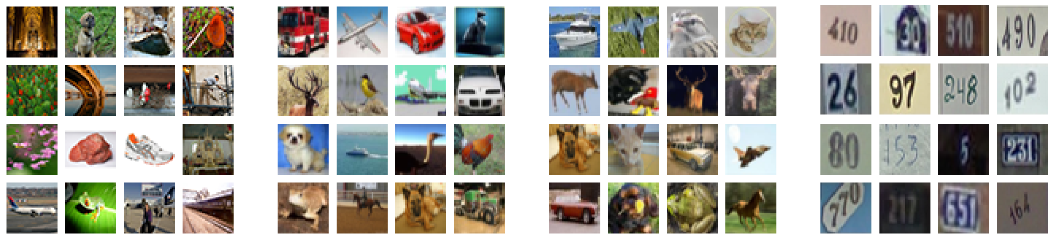
Algorithm 2 The proposed Greedy AutoAugment algorithm.

```
Input: Dataset $\mathbf{X}$.
Output: Best augmentation policy functions $\hat{f}_1, \ldots, \hat{f}_d$.
 1: for all augmentation techniques $t$ = 1,..., $t_n$ do
 2:     for all magnitudes $m$ = 1, ..., $m_n$ do
 3:         set probability to 1;
 4:         $\mathbf{X} \rightarrow \mathbf{X}_{tr}, \mathbf{X}_{te}$;
 5:         $\hat{f} = f(t, p, m)$;
 6:         apply $\hat{f}$ on $\mathbf{X}_{tr}$;
 7:         store policy with its score on $\mathbf{X}_{te}$.
 8:     end for
 9:     store best policy
10: end for
11: for counter = 2,..., $k$ do
12:     $\hat{f}$ = best policy;
13:     for all augmentation techniques $t$ = 1,..., $t_n$ do
14:         for all magnitudes $m$ = 1, ..., $m_n$ do
15:             set probability to 1;
16:             $\mathbf{X} \rightarrow \mathbf{X}_{tr}, \mathbf{X}_{te}$;
17:             $\hat{f} = \hat{f} + f(t, p, m)$;
18:             apply $\hat{f}$ on $\mathbf{X}_{tr}$;
19:             store policy with its score on $\mathbf{X}_{te}$.
20:         end for
21:     end for
22:     store best policy
23: end for
24: select $\hat{f}_1, \ldots, \hat{f}_d$ from all policies which have highest scores;
25: return $\hat{f}_1, \ldots, \hat{f}_d$
```

which represents the probability of choosing a set S_i or the original data. The v_1 corresponds to selecting the original data and $v_2, \ldots, v_{d/\lambda}, v_{(d/\lambda)+1}$ correspond to the selection of $S_1, \ldots, S_{d/\lambda}$. To fill the values of $\vec{v}$, we use the Pareto Distribution [80], as follows,

$$\rightarrow v_i = \begin{cases} (\frac{1}{i})^{\alpha} & i > 1 \\ 1 & i \leq 1 \end{cases} \tag{3.2}$$

In this equation, a positive parameter (α) is used to assign the highest probability to v_1 and lowest probability to $v_{(d/\lambda)+1}$. When $i = 1$ the probability is one, and for $i > 1$ the probability is less than one but not zero. To choose the best set S_i or original data, we start from the rightmost element of $\vec{v}$ and go to the leftmost element of $\vec{v}$. Each of these elements has a chance to be selected based on their respective v_i values. After choosing the S_i, a policy



(a) (b) (c) (d)

Figure 3.3: Samples from real datasets used in our experiments: (a) Tiny ImageNet (b) CIFAR-10 (c) CIFAR-100 (d) SVHN.

is chosen randomly from the members of the set, with uniform distribution.

3.1.4 Results

In this section, we compare our method with current solutions. For this purpose, first, we show the accuracy results of our method compared to the other methods. Next, we provide a computational analysis of the overall augmentation process. The AutoAugment section uses state-of-the-art record-breaking networks also to test their method. This encompasses several architectural advancements, which increases RAM requirements in GPUs and require at least 1800 epochs for each test-case. Using such a vast requirement to test augmentation policies is actually not necessary and would limit us to analyze the methods thoroughly. Therefore, in order for the experiments to be in our available resources, we have used a smaller and reasonable infrastructure to test both our methods and the AutoAugment results in a uniform setting. According to the AutoAugment section, their method is transferable. Therefore, this change is fair and should not affect their method.

In these experiments, we use eleven different prominent ANN architectures. The DenseNet121 [104], GoogLeNet [102], MobileNet [119], MobileNetV2 [120], PreActResNet18 [121], ResNet18 [103], ResNeXt29 [122], ShuffleNetG2 [123], ShuffleNetV2 [124], and VGG [125] are used as prominent networks which contain both normal and lightweight networks. To implement these networks, we forked the implementations from [126]. The default settings of the network implementations are not changed. The networks accept 32×32 images and provide an output based on the number of classes.

In the experiments, we also use four real datasets, 1- Tiny ImageNet [127] includes 120000

Table 3.2: The result table for accuracy analysis. Abbreviations include: Manual = Manual Augmentation, GAutoAugment = Greedy AutoAugment.

Network	Manual				AutoAugment				GAutoAugment			
	Tiny	Cifar10	Cifar100	SVHN	Tiny	Cifar10	Cifar100	SVHN	Tiny	Cifar10	Cifar10	SVHN
DenseNet121	34.65	80.67	53.91	91.69	29.15	80.20	54.80	91.47	**37.75**	**83.36**	**59.32**	**93.26**
GoogLeNet	**33.78**	79.85	38.96	89.63	27.81	**79.92**	**53.28**	90.79	28.70	79.25	50.51	**91.44**
MobileNet	14.58	**68.50**	40.75	81.43	10.50	63.95	37.28	79.53	**17.38**	64.90	**43.09**	**88.55**
MobileNetV2	**21.66**	70.81	39.92	84.33	15.00	67.95	39.14	83.42	21.07	**73.67**	**44.92**	**88.17**
PreActResNet18	29.45	77.16	44.57	89.06	26.01	83.05	**53.28**	89.77	**32.60**	**83.43**	48.26	**93.60**
ResNet18	29.61	78.68	42.34	87.08	27.51	80.22	**55.29**	89.83	**35.84**	**80.62**	48.25	**93.92**
ResNeXt29	31.87	76.95	47.71	**88.57**	23.85	78.15	50.90	81.63	**35.04**	**81.87**	**56.92**	84.08
SENet18	29.18	78.66	42.16	90.17	25.63	**80.55**	**54.18**	91.71	**31.96**	79.90	51.69	**93.07**
ShuffleNetG2	22.01	71.67	**48.93**	85.54	17.27	70.58	38.40	81.05	**27.72**	**74.56**	48.85	**91.14**
ShuffleNetV2	24.89	72.95	49.26	87.44	20.50	71.25	46.44	86.18	**27.10**	**75.27**	**53.19**	**91.88**
VGG	21.03	76.66	**43.41**	90.07	16.73	77.60	42.15	88.61	**23.16**	**80.75**	43.37	**93.06**

natural images in 200 classes with each class having a training set of 500 images a test set of 50 images along with 50 validation images. 2,3- CIFAR-10 and CIFAR-100 datasets [128], both containing 60000 images of size 32×32 in 10 and 100 classes respectively. 4- SVHN [129] which contains over 600000 images of real-world images of digits $0 - 9$. These selected datasets are used for three main reasons. First, while they are complex datasets, they have a reasonable number of images and features, which makes working with them with our available computational resources feasible. Second, they are well-known datasets with known and predictable results on a variety of ANN architectures. Third, they are compatible with the official experiments of AutoAugment section [130].

The proposed method can be used with all architectures and all datasets with reasonable training size. For training, we have two completely separate steps, 1-finding policies, 2-applying those policies. For finding policies, we use Algorithm 1, and then we perform normal training with the new policies. The default learning rate for networks in [126] is 0.1. We use the same learning rate throughout the experiments for finding policies and training networks. The number of epochs for all of the training scenarios was 200. To find the best policies, we need to create child networks. The child networks and training networks share the same infrastructures. To obtain the accuracies from child networks, we divided training data into two parts. The training part and the testing part. For Tiny Imagenet and SVHN, 5000 images are used for testing. For CIFAR-10 and CIFAR-100, 2500 images are used for testing. The images which are not used for testing part are used for training part. All of the images are selected randomly with i.i.d. distribution. The α value for Pareto Distribution was always 2. We also used $d = 25$ and $\lambda = 5$.

Accuracy

In this section, we test the accuracy of our proposed method using four different datasets, 1- Tiny ImageNet, 2- CIFAR-10, 3- CIFAR-100, and 4- SVHN. The results are shown in Table 3.4. In this table, the ”Manual” section stands for the images with common augmentation methods. These techniques include zero padding, cropping, random-flip, and cutout. To prevent probable errors, the same source code released in AutoAugment for augmentation techniques is also used for manual augmentation [130]. This helps to provide a fair environment for all methods. The only extra pre-processing step that we used is for resizing Tiny Imagenet from 64×64 to 32×32. This helped us to use the same infrastructure for all datasets without having adverse effects on the experiments. The values in the table are the average results from five trials.

The section ”AutoAugment” stands for the AutoAugment method. For CIFAR-10, CIFAR-100, and SVHN, we use the same policies that are found from AutoAugment method. For Tiny ImageNet, we use the policies that are found for ImageNet dataset. Because Tiny ImageNet is a subset of ImageNet, it can test the generalization of the AutoAugment method. The section ”GAutoAugment” stands for Greedy AutoAugment, which is the proposed method. In our method, we suggest a specific searching process for each scenario to find the best policies. This is possible because (as we will see in the next section), our search method is computationally much more efficient than AutoAugment method.

For Tiny ImageNet, as we can see, the policies from AutoAugment are not effective when they are applied to a smaller subset of the same dataset. All of the networks had worse results compared to the Manual augmentation. Overall, the AutoAugment reduced the accuracy with 52.73% compared to Manual augmentation. Comparatively, our method increased accuracies for nine out of eleven available networks. The highest increase of accuracy is for ResNet18, with 6.23% higher accuracy. The least increase of accuracy is for VGG with 2.12% higher accuracy. Overall, compared to the AutoAugment, the proposed method provided 78.34% higher accuracy for eleven networks. Respectively, compared to the Manual augmentation, the proposed method provided 25.60% higher accuracy for eleven networks.

For CIFAR-10, the transition is better for AutoAugment policies. From eleven networks, six networks had better results with at most 5.89% better accuracy and at least 0.06% better accuracy than the manual augmentation. On the other hand, the manual augmentation showed better results for five networks with at most 4.54% better accuracy for MobileNet and at least 0.46% better accuracy in DenseNet121. The overall increase of the accuracy was 0.86% in favor of the AutoAugment. Comparatively, our method increased the accuracies for nine networks, with at most 6.27% better accuracy for PreActResNet18 and at least 1.24% better accuracy for SENet18. Overall, we could increase the accuracies, 25.02% better than manual augmentation and 24.15% better than AutoAugment.

The results for CIFAR-100 show that when policies from AutoAugment were applied to original images, the accuracies could improve for six networks. The best increase of the accuracy is for GoogLeNet with at most 7.49% better accuracy, and the least accuracy is for DenseNet121 with at least 0.91 better accuracy. The overall increase of the accuracy was 1.04% for AutoAugment compared to the manual augmentation. When the policies from the proposed method are applied to the original network, we could improve the results for nine networks. The accuracy could be up to 11.54% and down to 2.34% better than the original images. The overall increase of the accuracy for the proposed method is 56.46% compared to the manual augmentation. The overall increase of the accuracy for all of the networks is 55.422% for the proposed method compared to the AutoAugment.

For SVHN, the AutoAugment provide higher accuracies for four networks when it is compared to the manual augmentation. The highest accuracy is for ResNet18 with 2.75% better accuracy and the least higher accuracy is 0.70% for PreActResNet18. For the six other networks, the manual augmentation was, on average 2.45% better than AutoAugment. It was at least 0.21% and at most 6.93% better than AutoAugment. Overall, manual augmentation provided 11.00% better cumulative results compared to the AutoAugentation. Comparatively, the proposed method provided better accuracies for ten networks compared to the manual augmentation. The highest and lowest accuracies were 7.11%, 1.57% better than manual augmentation. Overall, the cumulative accuracies are 37.16%, and 48.16% better than manual augmentation, and AutoAugment. The better accuracy comes from the fact that SVHN needed policies that are in further layers of the search space, and this could only be achieved with Greedy AutoAugment.

Computational Analysis

As described in Section 3, the search space is reduced from the exponential growth of base $(t_n \times p_n \times m_n)$ with the value of ℓ to the linear growth of $(t_n \times m_n)$ with the value of k. We already saw that this approach is effective in improving the accuracy of the networks. The important question is now how much computational resources we needed to perform the search. To allocate the resources, we are concerned about two aspects, 1-the search space, 2-real resources allocated for the searching process. We separate this two processess because the algorithms may not use all of the trials is searching spaces.

Table 3.3: The result table for comparing the search space between our method and AutoAugment.

Layers	AutoAugment	GAutoAugment	Comparison
$\ell = 1$	2200^1	200	11.0
$\ell = 2$	2200^2	4200	1152.3
$\ell = 3$	2200^3	8200	1298536.5
$\ell = 4$	2200^4	12200	1920131147.5

The comparison between the two spaces is shown in Table 3.5. This table has three columns. The first column shows the number of the layers, which is ranged from $\ell = 1$ to $\ell = 4$. The second column is for the search space for the AutoAugment algorithm. This shows the maximum number of possible selections that AutoAugment can choose. As we can see, the values for each row exponentially increases with the increase of the ℓ from a base of 2200. The 2200 is from calculating $(t_n \times p_n \times m_n)$. The third column represents the search space for Greedy AutoAugment. To calculate the search space, we use the (2). In this case, if each layer is fully explored, the number of possible trials is only 4000. The last column shows the comparison between column two and column three. The values show that the number of times the search space of the proposed method is smaller than the search space of AutoAugment.

For the real experiments, AutoAugment only searches for the 2-layer search space. This means that AutoAugment can have 2200^2 possible trials. Because searching all of the space for 2200^2 trials is impractical, the AutoAugment uses a sample of 15000 child networks. Comparatively, our method, in each step, explores 200 trials. The number of steps is determined by k. In these experiments, we used $k = 5$, which gives us 1000 trials. This number is used to limit the search within our computational resources. Also, AutoAugment uses 120 epochs to evaluate the accuracies of child networks. The number of epochs used for our child networks was only 5 epochs. Since the exact infrastructure for different datasets is not known for AutoAugment, and child networks are interchangeable, we consider the child networks to have the same efficiency. Therefore, overall we had $(15000 \times 120) \div (1000 \times 5) = 360$ less computational requirement. Note that with CUDA parallel programming [131, 132, 133, 134], the entire sub-policy layers can be explored in parallel at once, making the gained performance even better.

3.2 Greedy AutoAugment for Small Datasets

It has been shown that neural networks can perform extremely well in applications with large and complex datasets such as for different types of computer vision tasks [135]. An important constraint for these networks is the required large number of labeled data for the training process. This limitation could be problematic for many other application fields, which can only guarantee a small number of training data points [136]. The problem is more severe with recent deep neural network architectures that have many more hidden layers and parameters which need more data compared to previous generations of neural networks [103, 104, 137]. This is a few-shot learning problem where it is expected to obtain higher accuracy with small datasets. The smallness of the training data is a relative term in which it is desired to maintain or achieve higher accuracy with datasets that are as small as possible [138].

Few-shot learning is a methodology which deals with small datasets and takes many learning forms based on available data. If we can only rely on the available training resources, it is supervised. If unlabeled data or some helper classes are available, it could take the form of semi-supervised or reinforcement learning [139, 138]. In imbalanced learning, the few-shot can be used to take one part of the data as prior knowledge [140]. In transfer learning, the source domain contains different classes which are not available in the target domains [136]. In this case, few-shot learning is used for the target domain. Another popular form of the few-shot learning is meta-learning methodology, where application learners use meta-learners as prior knowledge [141].

The few-shot learning for classification problems is usually called n-shot learning, where n is the number of points in each class [142]. The most popular forms of n-shot learning are 1-shot and 5-shot learnings. These methods are divided into three categories [143]. The metric learning methods find some form of similarity space that is effective for n-shot datasets [1, 144, 145]. The memory learning methods train to learn how to store and retrieve memories with experiences to generalize unseen tasks [2, 146, 147]. The gradient descent methods rely on meta-learners that adapt to specific base-learners for n-shot datasets [148, 149, 150, 151, 152, 143]. Most of these methods can combine their infrastructures with data augmentations to further improve their current performance. Therefore, an improvement in data augmentation for small datasets can potentially improve current and future n-shot methods.

In this section, we show that Greedy AutoAugment, is in compliance with n-shot learning, where we need to use all available training data to see the real effects of sub-policies. In training, with the proposed method, the number of augmentations is much more than normal training, which helps us to collect as much information from the data as possible. The experiments show the effectiveness of the proposed method for both resource allocations and accuracy.

3.2.1 Greedy Breadth-First Search AutoAugment

In this section, we propose Greedy Breadth-First Search AutoAugment (GBFSAug) as a greedy algorithm to find the best augmentation techniques and apply them for training. For this purpose, we need two separate steps, 1- the searching process, which finds the best augmentations based on the scoring criterion defined in the general model. 2- the training process, which trains the data with the best augmentations, which are found in the first step. In the searching and training process for finding the best augmentation techniques, and performing final classification, the infrastructure is used as a black box. In this regard, the future neural networks that need augmentations can also be used for the proposed method. The design decisions for both steps specifically are created to take n-shot learning into account.

Let us consider each image as a multivariate data point $\vec{p_i}$. We formulate the classification problem as a collection of testing data points X_{te} which need to be classified into k number of clusters $c_1, \ldots, c_k$. The most effective way for such categorization is to use training data points X_{tr} as a knowledge set. If X_{tr} is available, it is possible to use neural network architectures to move $\vec{p_i}$s from their original space into a new space that separates c_is as much as possible.

Obviously, if X_{tr} has the same distribution with X_{te}, the training of the X_{tr} leads to a better generalization of X_{te}.

The training process of deep neural networks typically require a high number of $\vec{p_i}$s, where $\vec{p_i}s \in X_{tr}$. The problem of n-shot learning for these scenarios is that there is not enough members of X_{tr} to train for X_{te}. More specifically, n-shot learning is defined for n-shot k-class scenarios, where n specifies the number of $\vec{p_i}s \in X_{tr}$, for each c_i. For instance, a 1-shot 5-class scenario, indicates that each $c_i, c_i \in c_1, \ldots, c_5$ has one $\vec{p_i}, \vec{p_i} \in X_{tr}$. In this regard, the number of the points in X_{tr} is five. The problems that we are concerned in this section are for scenarios where $n \leq 100$. The most popular number for k in n-shot learning is 5 which leads to X_{tr} with ≤ 500 members.

In the searching process, to find the best augmentation policies, we perform many trials and select the policies that provided us with the best trials. To select the best policies, it is mandatory to define a scoring criterion for evaluation of the trials. A simple approach to this problem is to divide X_{tr} into two parts, $X_{tr} \rightarrow X'_{tr}, X_{val}$. Respectively, the X'_{tr} is the new training set, which is used by a neural network to generalize for the X_{val} as a validation set. While this approach would work for X_{tr}s with a high number of the points, it is not suitable in n-shot learning where every data point is a valuable asset and excluding a X_{val} set from training would have noticeable impacts for the final result. A better approach for scoring is K-Fold validation [153], which uses all of the members of X_{tr} for getting a score.

Let us divide the members of X_{tr} into k number of subsets. In our design, we assume the k, which is used in K-Fold algorithm to be equal to the number of classes. For each subset $i, i \in 1 \ldots k$, we define X^j_{val}, as a validation set that contains all of the members of subset j. Similarly, X^{-j}_{tr} is the training set that contains all of the points that are not a part of subset j. All of the subsets should be almost equal in size. Let us define N to indicate the functionality of a neural network. The policy function $\hat{f}_i$ can include several sub-policy functions (fs) and is used as the functionality of applying an augmentation policy for trial i, we have,

$$v_i = \frac{1}{k} \sum_{i=1}^{k} N\left(\hat{f}_i(X^{-j}_{tr})\right) \rightarrow L(y_j, N(X^j_{val})) \tag{3.3}$$

The training of X^{-j}_{tr}, includes applying of the augmentation policy function $\hat{f}_i$ with neural network N. The calculation of loss function (L) provides the real values for the evaluation of each trial score, v_i. To get L, we use X^i_{val} with N and without applying $\hat{f}_i$. The final value for each trial (v_i) is an average of all the loss functions.

To clarify the overall process, we present Figure 4.4. As we can see, the classification includes three main steps. In the search step, the training set X_{tr} is given, which includes n number of the points. There is x number of $\hat{f}_i$s, which should be applied to the training set (search space). Next, we need to score each $\hat{f}_i$ in which the (1) is used to get the scores. As we can see in the score section, there is d number of v_i values, which are the best values from x trials. The third step is for training, that divides the new training set into $b_1, \ldots, b_y$ batches. Each batch can contain either original data points or augmented datapoints.

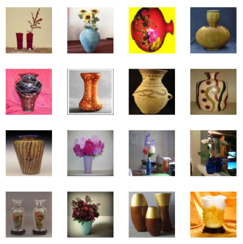
In the searching phase, we use Greedy AutuAugment which is covered in the previous section. In the training phase, we apply all of the d number of augmentations to increase the size of the training data. For the n-shot learning, it is easier to apply augmentations as a pre-processing step instead of doing them for each epoch. The goal is that the policies that have a higher score to be selected. On the other hand, it is crucial to increase the diversity of X_{tr} as much as possible. For this purpose, we use (3).

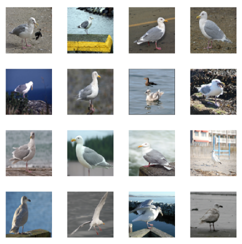
$$X'_{tr} = X_{tr} + \sum_{i=1}^{d} \hat{f}_i(S_i) \tag{3.4}$$

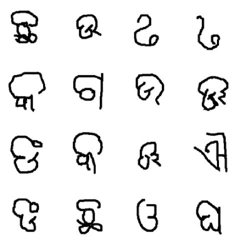
In this notation, the X_{tr} is the original training set. For all of the d augmentation policies, we select a set S_i from training set X_{tr} ($S_i \in X_{tr}$). The augmented training data X'_{tr} is the sum of X_{tr} plus the the applied augmented policies $\hat{f}_i s$ on samples S_is ($\hat{f}_i(S_i)$s). For final training, we repeat this process to get as much data as desired and shuffle them before training. We suggest the d value be much higher than what we would use for normal datasets. This allows us to extract more information from the limited number of data points.

3.2.2 Results

In this section, we provide our experimental results to analyze the effectiveness of the proposed method. In the experiments, we used five prominent n-shot learning datasets. Samples of these datasets are shown in Figure 4.2. For numerical values of the search space, we follow


(a)


(b)


(c)

(d)

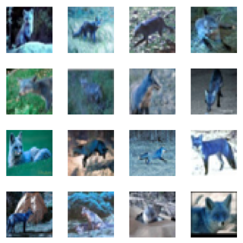

(e)

Figure 3.4: Samples from real datasets used in our experiments, the images are randomly selected from: (a) miniImageNet [1] (b) CUB-200-2011 [4],(c) Ominiglot [3],(d) Flower102, and FC100 [2].

the discretization setup, which is introduced in AutoAugment. The number of values for t_n, p_n and m_n, are in the order $20, 11$ and 10. In all of the scenarios for the proposed method, we use $k = 21$, which means the proposed method performs 4200 trials. We use $d = 1000$ which means that 1000 augmentation policies from 4200 trials are selected. The S_i value is 5, for all n-shot scenarios. The data augmentation is used as a preprocessing step that augments the data to 60000 points.

The experiments are divided between five main parts. The first and second sections demonstrate that our method extracts information properly from a few data. For experiments, we compare our method with supervised solutions, which are directly related to the proposed method. In the next two sections, we demonstrate that our method integrates properly with n-shot methods. For experiments, we integrate the proposed method with Siamese and MAML.

In the end, we compare our method with state-of-the-art n-shot methods. The purpose is to see that with the improvements of the proposed method, when fully supervised learning starts to become a viable solution. Our experiments show that the n-shot methods are effective in their approach and simple classifications, even with auto augmentation require more shots to provide better results. Therefore, for the best outcome, the integration of our proposed method with n-shot methods is recommended.

3.2.3 Analysis for number of shots

In this section, we perform a comparison between our method and the current solutions which provide augmentation for neural networks. The goal is to analyze the method for different number of shots. For a comprehensive analysis, we use 1-shot to 100-shot settings, with the three commonly used datasets in n-shot learning. The first dataset is miniImageNet [1] that is proposed for n-shot learning. This dataset is a subset of ImageNet [154] that contains 84×84 colored images with samples of 600 in 100 classes. As it has been suggested in [149, 148, 152, 155, 156], the samples are divided into 64, 16, and 20 classes respectively for meta-training, meta-validation, and meta-test. The second dataset is FC100 [128] which is a subset of CIFAR-100 [128]. This dataset includes 32×32 images with 600 samples and 100 classes. As suggested in [143], meta-training data are from 60 classes that belong to 12 super-classes. Respectively, the meta-validation and meta-test, contain 20 classes which belong to 4 super-classes. The third dataset is Ominiglot [3], in which we used each language as a separate class and not the characters. In this dataset, characters are very similar, and having multiple shots for each character would lead to complete accuracy in most cases. Separating the dataset with languages instead of characters helped us to alleviate this problem. More importantly, we needed up to 100 samples from each class, which was not available in the original form of Omniglot.

We compare our method with three other approaches. 1- Networks without using augmentations (NoAugment). 2- networks with the most common augmentations (CommonAug). These techniques are zero padding, cropping, random-flip, and cutout [113]. 3- The AutoAugment method, as described in [113]. All of the four methods use ResNet-18 for training and testing. To perform the experiments, we use PyTorch 1.0.1. The standard implementation of ResNet [103] from TorchVision 0.2.2 repository is used [157]. For the basic setup, we use the exact parameters as described in the default values of [158]. Some of these parameters are: learning-rate = 0.1, start-epoch = 0, end-epoch = 90, momentum = 0.9 and weight-decay =le-4. All of the images are resized from their original input size to 224. This allowed us to make the network compatible with all scenarios similarly. The result is an average of 3 repeated random

Table 3.4: The result table for accuracy analysis of our method compared to supervised methods on miniImageNet [1], FC100 [2], and Ominiglot [3]. Numbers are the percentages of obtained accuracies. The goal is to analyse the method for different shots.

miniImageNet	1-shot	5-shot	10-shot	25-shot	50-shot	75-shot	100-shot
NoAugment	23.00	30.33	28.00	32.66	36.33	51.00	38.00
CommonAug	22.66	29.33	35.66	34.66	39.33	35.66	49.66
AutoAugment	19.66	38.00	31.33	25.66	29.33	25.00	24.00
GBFSAug	**34.66**	**43.66**	**44.00**	**64.00**	**59.33**	**58.66**	**74.00**
FC100	1-shot	5-shot	10-shot	25-shot	50-shot	75-shot	100-shot
NoAugment	19.66	57.33	44.00	34.66	40.00	31.33	46.33
CommonAug	15.33	46.33	41.33	35.66	32.33	52.00	41.66
AutoAugment	15.33	62.66	34.00	41.00	37.33	31.33	36.66
GBFSAug	**21.66**	**71.33**	**52.66**	**56.00**	**65.66**	**62.33**	**62.66**
Ominiglot	1-shot	5-shot	10-shot	25-shot	50-shot	75-shot	100-shot
NoAugment	25.00	49.00	52.66	60.33	67.00	75.33	74.66
CommonAug	26.33	39.00	40.66	48.66	47.66	60.66	62.33
AutoAugment	26.33	36.66	41.00	42.66	47.66	58.00	62.66
GBFSAug	**29.00**	**59.00**	**64.33**	**85.66**	**87.66**	**81.00**	**90.33**

trials without using seeds.

All of the experiments are done with a 5-class n-shot setting. All three datasets are designed specifically for transfer learning in mind. In our method, we are concerned about the supervised setting where X_{tr} has the same distribution with X_{te}. For this reason, we do not use the train and validation classes from the datasets. In the experiments, at most, we use 100 samples for training. With this assumption, we use the testing classes to create both X_{tr} and X_{te} sets. More specifically, 5 classes are chosen randomly from available options in the testing classes. For training, based on the value of n in n-shot, we select training sets randomly. For instance, in 10-shot testing, we select 10 samples from each class. To calculate the scoring criterion (see Section **??**), the X_{tr}^{-i} and X_{val}^{i} are created from X_{tr}. To create X_{te}, 300 samples that are not selected for X_{tr} are randomly selected for final testing.

The results are shown in Table 3.4. As we can see, the table is divided into three sections for miniImageNet, FC100, and Omniglot datasets. The range of the shots in n-shot scenarios is from 1 to 100. The highest accuracies for each shot are shown with bold numbers. For miniImageNet the proposed method is better than the other methods in all of the 7 scenarios. The highest increase of the accuracy is for 100-shot scenario compared to the AutoAugment with 50% more accuracy. The lowest increase of the accuracy is for 25-shot scenario compared to the no augmentations (NoAugment) with 7.66% more accuracy. The proposed method, on average, has $19.85\%, 18.76\%$, and 26.47% higher accuracies than NoAugment, AutoAugment, and CommonAug.

The maximum increase of the accuracy for FC100 is 33.33% for the 50-shot scenario compared to the CommonAug. The least increase in accuracy is 2.0%. On average, the increase of the accuracies compared to AutoAugment, NoAugment, and CommonAug are in order $16.99\%, 18.23\%$, and 19.14% better. Respectively, the maximum increase of the accuracy for Ominiglot is 43.00% for the 25-shot scenario compared to the AutoAugment. The least increase in accuracy is 2.67%. On average, the increase of the accuracies compared to AutoAugment, NoAugment, and CommonAug are in order $13.28\%, 24.52\%$, and 26.00% better.

For the other three algorithms, the AutoAugment had the worst results. The NoAugment was usually better than CommonAug. This is specifically true for FC100 and Ominiglot. The CommonAug had the best results for miniImageNet. In the table, a noticeable pattern is the apparent fluctuations among specific shots. This means that, in some cases, lower shots could provide higher accuracies compared to the higher shots. Such patterns should be expected for n-shot settings where, in rare cases, the generalization of some small batches could be very high. However, in general, such patterns are not reliable, and higher shots provide better accuracies.

3.2.4 Analysis for number of classifications

To analyze the proposed method for different number of classifications (e.g., n-way), we apply our method on two additional datasets, CUB-200-2011 [4] and Flower102 [5]. These datasets are also mainly used in few-shot learning and have more than 100 classes, which makes them useful for our experiments. The results are for n-class 5-shot settings. For these experiments, we follow the common practices for n-shot classification. For each dataset, we take five samples out of the testing class, search for augmentations for those five samples, train the network, and then test the dataset. These results are an average of 3 trials. All of the images have been converted to an 'RGB' image with the rescaling to 254×254 resolution.

The results are shown in Table 3.5. As should be expected, since n-shot is fixated to 5,

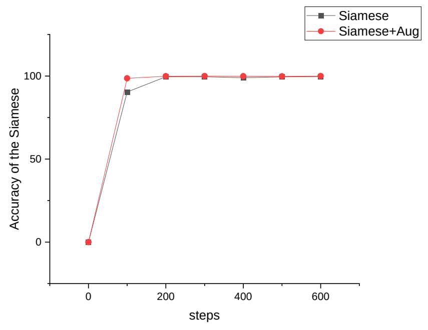
Table 3.5: The result table for accuracy analysis of our method compared to supervised methods on CUB-200-2011 [4] and Flower102 [5]. Numbers are the percentages of obtained accuracies. The goal is to analyse the method for different classifications.

CUB-200-2011	2-way	5-way	10-way	25-way	50-way	75-way	100-way
NoAugment	56.66	38.66	24.00	11.66	4.00	8.00	5.00
CommonAug	**83.33**	41.33	16.00	7.33	7.00	6.66	4.00
AutoAugment	60.00	34.66	17.33	6.66	4.33	2.66	2.33
GBFSAug	**83.33**	**49.33**	**36.00**	**17.66**	13.33	**13.33**	**9.00**
Flower102	2-way	5-way	10-way	25-way	50-way	75-way	100-way
NoAugment	76.66	62.66	41.33	43.00	28.00	32.66	30.00
CommonAug	76.66	78.66	44.66	41.66	30.00	30.33	31.00
AutoAugment	76.66	60.00	33.66	34.33	24.66	18.33	22.33
GBFSAug	**80.00**	**86.66**	**75.33**	**56.66**	**45.66**	**49.33**	**47.66**

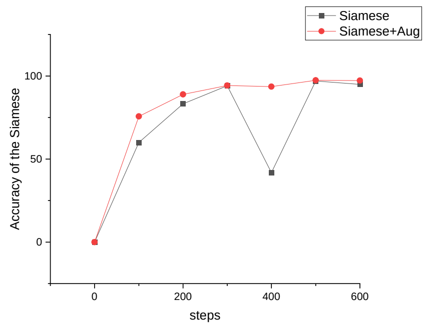
as we increase the n-way setting, classification becomes less accurate. The outcomes in the table show that the proposed method can reliably improve the accuracy of both datasets for different classification numbers. More specifically, for the CUB-200-2011, the highest increase of the accuracy for GBFSAug is for 2-way setting with 26.67% better accuracy compared to the AutoAugment. The averages of higher accuracies are 10.57%, 8.04%, and 13.43% compared to NoAugment, CommonAug, and AutoAugment. For the Flower102, the highest increase of the accuracy is 41.67% in 10-way compared to the AutoAugment. The lowest increase of the accuracy is 3.34% in 2-way. The averages of higher accuracies are 18.14%, 15.47%, and 24.47% compared to NoAugment, CommonAug, and AutoAugment.

3.2.5 Case study on Siamese

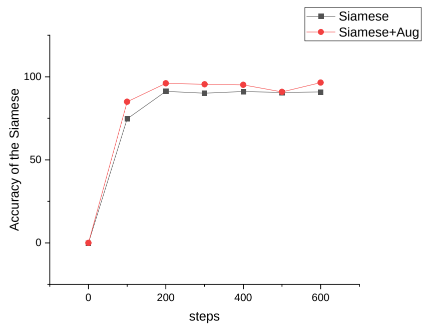
The proposed method is similar to plain augmentation techniques. This means that the underlying network is considered as a black box that allows other methods to use it on different architectures and for different goals. An important argument immediately arises that how much augmentation can help current n-shot learning methods. The main obstacle for answering this question is that the methods that use augmentation may use it for many applications, which cannot be covered in the experiments. One methodology that works for one application may not work for others. For instance, depending on the application: Should we use augmentation for the training set or the test set? Should we increase the size of the data, or let them be integrated with the training sets? One part of the algorithm should be augmented or all of it?



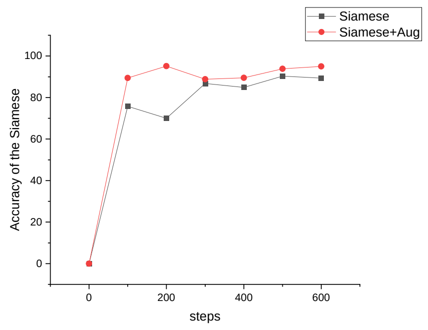
(a) Ominiglot



(b) CUB-200-2011



(c) Flower102



(d) miniImageNet

Figure 3.5: Augmentation policies found from the proposed method are applied to the training set for Siamese neural networks for one-shot image recognition. The augmentation can increase the accuracy of the algorithm for data sets Ominiglot, CUB-200-2011, Flower102, and miniImageNet.

And many other questions that may arise with each method. That is why we do not want to impose the idea that this method needs to use a specific architecture to work. However, to show the effectiveness of the proposed method, it is still useful to integrate the method with n-shot learning methods. These methodologies are only for case-study and not a blueprint for all methods.

For the first integration, we use the Siamese network for 1-shot image recognition [159] as a prominent and influential method, which is easy to integrate and simple to understand. This network belongs to the metric learning family where it can recognize whether two points are in the same class or not. In the design description of [159], the available 1-shot data is not used for training. With the proposed method, it is easy to multiplicate the augmentations of the available 1-shot data to create new classes in training. This can boost the learning ability of the network, which can eventually lead to better distinguish the test cases. For the implementation, we use [160]. The default values from the experiments are not changed. For the experiments, we use two datasets from Section 3.2.3 and two datasets from Section 3.2.4. The only modification on datasets is a conversion from their original resolution to 105×105, and also converting the 3 channel RGB images to the single channel. These preprocessing steps allowed us to use the same architecture form [160] for all datasets.

For each class in the test, we used the one available data and created a separate class with an equal number of images that are presented in other classes. To create the new classes, the first 100 augmentations were selected and used randomly. The newly created augmented classes were treated equally to the original training data. For testing, the same available data which were used to create new classes for training, were used for distinguishing between pairs of points. The pairs could be between the same class or other classes that are in the testing sets. In each 100 steps of the training, 2000 testing pairs were selected to test the algorithms.

The results for Omniglot is shown in Figure 3.5a. As we can see, the specific structure that is used in Section 3.2.3 boosted the results for Siamese network on Ominiglot from their original reported accuracy. The results, started from 90.35% in step 100 to 99.60% in step 600. The results of the proposed method, when combined with the original network, could help the network to increase the accuracy for upto 8.25%. For the results of CUB-200-2011 (Figure 3.5b), the Siamese network started from 59.99% in step 100, and reached to the accuracy of 94.95% in step 600. A big fluctuation occurred in step 400. When it was combined with the proposed method, the network started from 75.65% in step 100, and reached to the accuracy of 97.2% in step 600. The results for Flower102 (Figure 3.5c), started from 74.7% in step 100 and reached to 90.85% in step 600. When it was combined with the proposed method, the accuracy was 84.95% and reached to 96.5% accuracy in step 600. For miniImageNet (Figure 3.5d), the accuracy for the proposed method was at most 95.15, which is 4.85% higher than what could be achieved with the original Siamese network. These results show the effectiveness of the proposed method for the type of problem that Siamese networks try to solve. Similarly, other n-shot learning methods can also find the aspects of their algorithm, which can be improved from using the proposed method and combine them to boost their outcomes.

3.2.6 Case study on MAML

To further examine the effectiveness of the proposed method, we integrate the greedy augmentation with MAML. Similar to the Siame, MAML is a well-known algorithm with a simple structure. For MAML, there are several approaches that can take advantage of augmentation. The approach that we take is to apply augmentation to both query set and support set. In this way, the meta-learner can more conveniently learn the training data and use augmentations to better generalize its network. In the experiments, we used the implementation in [161]. The configuration was a standard 5-class 5-shot training dataset with miniImagenet. The exact policies that we have found in the main experiments are used for data augmentation.

The results are shown in Figure 3.6. As we can see, without overhead and with simple modifications, we could constantly increase the accuracy of the MAML network. In step 1, both methods have almost the same accuracy. The MAML has an accuracy of 27.69%, and augmented MAML has an accuracy of 27.83%. In step 2500, the MAML has an accuracy of 45.58%, and augmentation MAML increased its accuracy to 50.73%. The difference of accuracies in steps, $5000, 7500, 10000, 12500$, and 15000, are in order 7.52%, 5.95%, 6.2%, 6.47%, and 4.15%. The highest increase of accuracy is for step 5000, with 7.52% higher accuracy. If we exclude the first step, on average, the increase in accuracy is 5.9%.

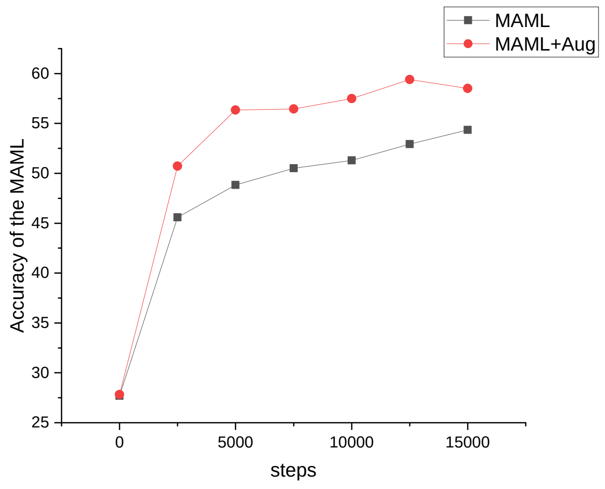
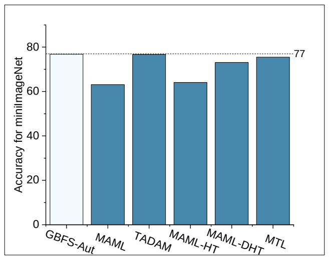


Figure 3.6: Augmentation policies found from the proposed method are applied to both support set and query set in the training phase for MAML algorithm. The augmentation increases the accuracy of the algorithm.

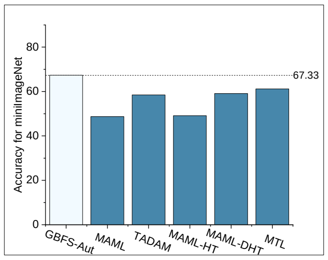
3.2.7 Comparison with n-Shot Methods

As mentioned, the proposed method is not designed to replace current solutions of n-shot learning. Instead, it has to be used as a complementary option to these networks. However, it is still useful to see that with the improvement in accuracy, how much training data is needed to reasonably generalize X_{tr} for X_{te}. For this purpose, we compare our method with state-of-the-art, n-shot solutions. These benchmark methods are useful because they are reaching this level of accuracy without a large number of training data points. In the following, we increase the number of the shots until classifiers are at least as good as these methods.

The n-shot learning methods are optimized to be very efficient for 1-shot and 5-shot scenarios. Among several options, we present five best methods which provide the highest accuracies [143]. The best methods in this category are TADAM [2], MAML , MAML-HT, MAML-DHT [149], and MTL [143]. For our method, we use the same training setup from Section 3.2.3. For the five n-shot methods, the exact unified infrastructure from [143] which itself extends from [162, 148, 2, 149, 1, 156, 146, 155] is used. Accordingly, the exact environmental setup from [143, 163] should be used. We did not change any settings and only replicated the results to reconfirm the reported accuracies. For the datasets we used, miniImagenet and FC100, which are also used in the original [143, 163] section. For the results of GBFSAug, we performed

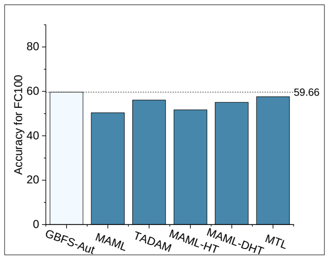


(a) 100-shot vs 5-shot

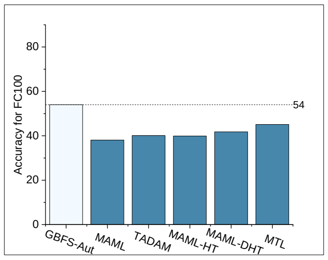


(b) 50-shot vs 1-shot

Figure 3.7: The result of our method applied on a supervised network compared to 5 benchmark n-shot algorithms on miniImageNet [1].



(a) 75-shot vs 5-shot



(b) 25-shot vs 1-shot

Figure 3.8: The result of our method applied on a supervised network compared to 5 benchmark n-shot algorithms on FC100 [2].

new experiments and did not use the values of Table 3.4. However, they approximately confirm the results of the table.

First, we present the results for miniImageNet. In Figure 3.7a, the proposed method is compared with the 5-shot scenario. As we can see, compared to the TADAM, which is state-of-the-art in this scenario ($\approx 77\%$), we could reach the same level of accuracy with 100-shot training data. The other methods are in order MTL, MAML-DHT, MAML-HT, and MAML with accuracies of $75.5\%, 73.1\%, 64.1\%$, and 63.11%. In Figure 3.7b, the proposed method is compared with 1-shot scenario. In this case, the 50-shot training data could surpass the 1-shot methods with an accuracy of 67.33%. The closest method is MTL, with an accuracy of 61.2%. The other methods are MAML-DHT, TADAM, MAML-HT, and MAML with accuracies of $59.1\%, 58.5\%, 49.1\%, 48.7\%$, respectively.

In Figure 3.8, the plots represent the comparisons on FC100. This time, the proposed method could outperform the 5-shot scenario with 75-shot training datasets. The best method for 5-shot learning is MTL with 57.6%, which is closely followed by TADAM and MAML-DHT with $56.1\%, 55.1\%$ accuracies. Accordingly, the accuracy for 25-shot datasets was 54%, which is followed by MTL with 45.1% accuracy. This result is followed by MAML-DHT, TADAM, and MAML-HT with accuracies of $41.8\%, 40.1\%$, and 39.9%. The worst result is for MAML, with an accuracy of 38.1%.

The overall observation is that for 1-shot learning, 50-shot and upward training sets provide better results. Comparatively, for 5-shot learning, 100-shot and upward training sets provide better results. These results show minimal acceptable dataset size for supervised learning, which is approximately $50 - 100$-shot upward.

Chapter 4

Automated Unsupervised Learning

4.1 Meaningful Distance for Multivariate Clustering

4.1.1 Introduction

We can divide the clustering methods into two different groups, high-level clustering algorithms, and low-level clustering algorithms. The high-level clustering algorithms do not assume any specific type of input and can deal with a variety of difficult problems such as dealing with non-linearity in data points or finding new features from relationships among current features. These methods are helpful for clustering more complex data such as images. In this regard, dimensionality reduction techniques (feature selection/extraction) such as subspace clustering [164, 165, 166, 167] and, autoencoders [168] combined with deep networks [169], can be considered as high-level clustering algorithms. On the other hand, the low-level clustering algorithms only accept a specific type of input data which is known a priori [170]. This basic assumption is the most important part of low-level clustering algorithms which changes the fate of the clustering results dramatically [171].

While the low-level clustering algorithms are restrictive on their basic assumptions for input data, they are still useful for many applications such diffrent parts of network managements and medical data [172, 173, 174]. These algorithms usually have more tractable complexity and are often included to provide a complete clustering package for more complex methods. For instance, state-of-the-art high-level algorithms for clustering image data such as [175] incorporate spectral clustering [176] in their designs which needs K-Means [177] as a helper function. Low-level clustering algorithms are also useful when input data have a specific structure which led to their popularity for certain use-cases. However, there are still some fundamental problems related to low-level clustering algorithms which are underdeveloped in the current literature. An essential problem is that almost all of the current basic assumptions of input data assume prespecified patterns in multivariate space. In these specific input types, we may experience synchronization inconsistencies between univariate and multivariate spaces which can result in irregular scenarios.

To make the problem clear let's consider K-Means algorithm. The basic assumption of K-Means is the separation of points based on Multivariate Gaussian (MVG) distributions. To this end, a multivariate distance metric such as Euclidean distance is used to measure differences of points in high dimensional data. A simple strategy to generate features for K-Means can be the decomposition of points based on Univariate Gaussian (UVG) distributions in each feature. Intuitively, with this guideline, points support their groups in all of their features without considering relationships among features. Even in this case which is probably the most basic guideline for data generation, clustering may not work properly. For K-Means to work on this guideline, points in the same group which are in the same UVG distributions for all or most of their features should also create a separated MVG distribution. However, even small irregularities in the features can potentially lead to unwanted clusters. As a simple example consider Figure4.1a.

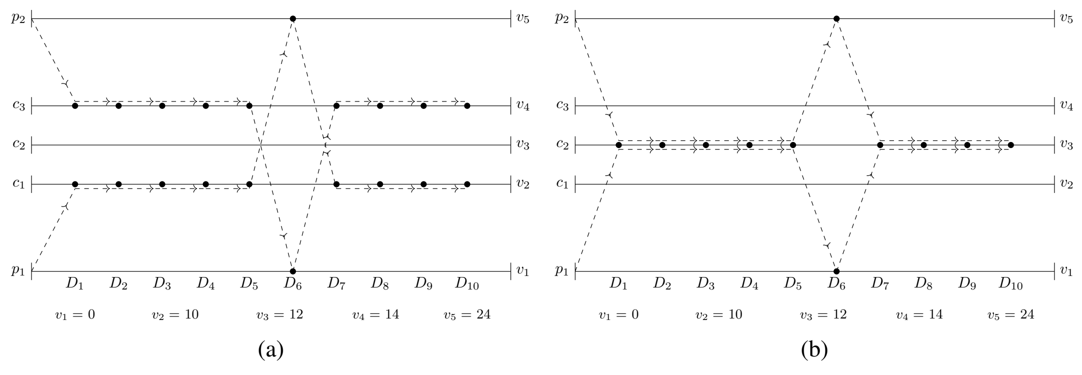

p_2
p_1
c_3
c_2
c_1
v_5
v_4
v_3
v_2
v_1
D_1 D_2 D_3 D_4 D_5 D_6 D_7 D_8 D_9 D_{10}
$v_1 = 0$ $v_2 = 10$ $v_3 = 12$ $v_4 = 14$ $v_5 = 24$
(a)
(b)

Figure 4.1: Separation of points for each feature may not lead to separation of points in multivariate space. Closeness of points in most features may not result to closeness of points in multivariate space.

In this example, we have two points p_1, p_2 that for each of their features (f_i), each point holds a completely separated value from the other point which is either $v_1 = 0$ or $v_5 = 24$. Let's consider three centers c_1, c_2, c_3 that take values $v_2 = 10, v_3 = 12, v_3 = 14$ for all their features. From these values, points p_1, p_2 are separated and belong to different clusters in all features. However, their Euclidean distances are equally close to the same cluster in multivariate space. Similarly, there are examples where the similarity in features may not result in the similarity of multivariate space. Figure4.1b shows an example where p_1, p_2 have nine similar features in c_2. However, because in one features they are dissimilar, they get separated in multivariate space ($p_1 \in c_1, p_2 \in c_2$). These two examples show that distances in individual features and distances in multivariate spaces are not necessarily synchronized. The situation can become worse as the number of points or clusters increase. The result is that based on the basic guidelines of low-level algorithms if points follow the guideline in multivariate space it does not mean they necessarily hold proper features. Likewise, if points follow the guideline in univariate space, they may not follow the guideline in multivariate space. The potential inaccuracies of data generation lead to inconsistencies and confusion for data analysis.

In this section, we propose a new clustering process that can replace current low-level clustering algorithms. In this regard, the first step is to streamline the underlying assumption of low-level clustering algorithms. Consequently, instead of expecting separation of points in multivariate space, we presume separation, for each feature individually. Based on this assumption, each point by default is responsible for defining its related cluster uniquely in each dimension. If there is a hidden relationship among features, we expect that such connection to be expressed as a new feature for the input data. We can interpret this primary assumption in two ways. First, it is yet another underlying guideline for input data that provides a more intuitive way for feature generation that can mitigate some synchronization problems between univariate and multivariate spaces. Second, as a new approach which potentially can help us to solve some problems connected with current low-level clustering algorithms.

A complete clustering process requires a methodology for separation of points based on the basic assumption of input data. This process is the second step of the proposed method which is also flexible, and separation is possible with different clustering algorithms. This includes clustering methods such as EM [178], variants of K-Means [179] or even Hierarchical Clustering [180]. However, to simplify the problem, throughout the section we focus on K-Means algorithm. As a result, the clustering in each dimension is similar to K-Means, and the assumption of input data is the separation in each dimension based on UVG distributions. Using this infrastructure, the ultimate goal of this section is to provide a comprehensive solution that addresses the following fundamental problem in low-level clustering algorithms.

The main problem is a claim that says clustering in multivariate space may become meaningless as the number of features grows. This problem is one of the most critical issues associated with the curse of dimensionality [181]. As described in Beyer's influential section [182, 183], considering that relative distance of farthest and nearest points converges to zero, distance, neighborhood, and proximity become less meaningful as dimensionality grows. This problem is called concentration effect with a significant consequence that even with correctly generated features, the clustering of the points can be meaningless. From other reports [184, 185, 186, 187] it is recorded that this is not a general rule and correlation of points play an important role in mitigating this effect. However, the problem is still not adequately addressed and can become quite problematic for clustering in high dimensions. We show that this problem can be solved based on our underlying assumption. In this regard, we define a new norm $\|.\|_c$ (read norm clustering) and subsequently Clustering Distance (CD) as a new distance metric which does not lose its meaning when dimensionality grows based on our underlying assumption.

4.1.2 System Model

Let us first define the necessary notations and overall representation of the clustering process. To represent multivariate data, we define samples as vector points $\mathrm{P} = \{\vec{p}_1, \ldots, \vec{p}_n\}$ where n is the number of points. The vector values are the features of each $\vec{p}_i$ and defined as $\mathrm{F} = \{f_1, \ldots, f_d\}$ where d is the maximum number of features. In this regard, each $\vec{p}_i$ is defined in $\mathbb{R}^d$ dimensional space and f_j is the value in a particular dimension D_j[1]. Let's suppose agent $\mathbf{A}$ is responsible to put members of the P into a specific number of clusters, e.g., k clusters. $\mathbf{A}$ can take the form of an end user of clustering software. It also can take the form of an upper-level algorithm which needs the clustering of its output data.

To make k clusters out of members of P, $\mathbf{A}$ needs a clustering algorithm. In clustering algorithms, based on different initialization settings, the membership of points to clusters can change dramatically. The most important initialization setting which is common in K-Means

[1]Throughout the section, indices i, j, e, h are indications of random members in sets unless told otherwise.

based algorithms is associative points (also called initial seeds)[2]. Let's take the results of any clustering algorithm with different associative points as $\mathrm{G} = \{g_1, \ldots, g_q\}$. In this regard, each g_i is a group that contains associative points $\mathrm{P}^{(\mathrm{k})} = \{\vec{\rho_1}, \ldots, \vec{\rho_k}\}$ which are the main elements for creation of clusters $\mathrm{C} = \{c_1, \ldots, c_k\}$. In this notation, $\vec{\rho_i}$ is an associative point and $\mathrm{P}^{(\mathrm{k})}$ is the set of all associative points for any group g_i. Accordingly, ρ_i is the initial seed that creates cluster c_i and C is the set of all clusters for any group G. We also denote $\mathrm{O} = \{\vec{o_1}, \ldots, \vec{o_k}\}$ in which $\vec{o_i}$ is the center of cluster c_i and O is the set of all centers.

4.1.3 Meaningful Clustering

In this section, we address a critical problem related to the curse of dimensionality which claims that distance metrics lose their meanings as the number of dimensions increases. First, we explain why this outcome exists. Next, we propose CD as a new distance metric which employs our basic assumption and does not fall prey to this effect. More specifically, we are concerned about the main result of the Bayers section [182] which is given in (4.1). Intuitively, let's take any vector point such as $\vec{o}_q$ (usually called query point) and consider $\vec{p}_i, \vec{p}_j$ as two points which have farthest distance (D_{max}) and nearest distance (D_{min}) to $\vec{o}_q$. With a higher number of dimensions, the relative distance of $D_{max} - D_{min}$ compared to the distance of D_{min} becomes very small.

$$\lim_{d\to\infty} var\left(\frac{\|X\_d\|}{\mathbb{E}[\|X\_d\|]}\right) = 0 \Longrightarrow \frac{D_{max}-D_{min}}{D_{min}} \to 0 \tag{4.1}$$

This phenomenon is called concentration effect which is presumed to be the primary source of the problem that makes clustering and classification in high dimensional data meaningless. This issue affects Minkowski distance form which is a generalization of three conventional distance metrics, Manhattan Distance, Chebyshev Distance and Euclidean Distance [188]. We can write the Minkowski distance as follows,

$$d(\vec{p_i}, \vec{p_j}) = \left(\sum_{e=1}^{d} (\vec{p_i}.f_e - \vec{p_i}.f_e)^p\right)^{\frac{1}{p}} \tag{4.2}$$

[2]In other algorithms such as EM, we can use another form of initialization settings. These settings can replace our definition of initial seeds if another separation algorithm is employed.

for p=1, the equation corresponds to the Manhattan Distance, for p=2 the equation corresponds to the Euclidean Distance and for $p = \infty$ it resembles the Chebyshev Distance.

The problem of the Minkowski distance is utilizing a stationary coordinate system which is not suitable for calculation of differences for clustering in high dimensions. The main obstacle is how origin which is a vector with values of zero (e.g., $[0, \ldots, 0]$) is used as a reference to provide a sense of similarity/dissimilarity with metric systems. While this stationary root is suitable for univariate space, it has detrimental effects when dimensionality grows. Since this point has 0 values (does not affect the calculations), it is omitted from (4.2). In this regard, if we define $\vec{o}_0$ as origin we can rewrite (4.2) in the following form,

$$d(\vec{p}_i, \vec{p}_j) = \left(\sum_{e=1}^{d} ((\vec{p}_i.f_e - \vec{o}_0.f_e) - (\vec{p}_i.f_e - \vec{o}_0.f_e))^p\right)^{\frac{1}{p}} \tag{4.3}$$

We use the notation "." to show the relationship between features and vectors. In this regard, $\vec{p_i}.f_e$ indicates the value of dimension e in vector point $\vec{p_i}$ and $\vec{o}_0.f_e$ indicates the value of dimension e in center $\vec{o}_0$. To mitigate the concentration effect, we want to replace the role of $\vec{o}_0$ with $\vec{o_i}$ which is the center of c_i. Unlike the origin, since $\vec{o_i}$ is the center of the c_i, it is not a stationary point and can take different values based on the members of the clusters. In this regard, the new reference traces center of the c_i and does not push all points unanimously apart. Instead, points that are a part of c_i are differentiated better compared to the points that are not a part of c_i. However, the feature values of $\vec{o_i}$ are not given a priori which creates a contradictory situation for calculation of (4.3). To clarify the problem, consider the underlying assumption of K-Means which requires points with the mixture of MVG distributions. To get $\vec{o_i}$ using K-Means, we need to cluster data in their multivariate forms and use distances with (4.2). This process requires the use of meaningless measures to provide meaningfulness for distance metrics which is contradictory.

To circumvent this limitation, we change the underlying assumption of the clustering data from MVG distributions to UVG distributions. In this way, we can enforce the separation of points in each $\mathbb{R}^1$ instead of $\mathbb{R}^d$. This approach results in the calculation of $\vec{o_i}$ in a meaningful way. Subsequently, we can define a new metric system and clustering process which guarantees meaningfulness of distance values and clustering results.

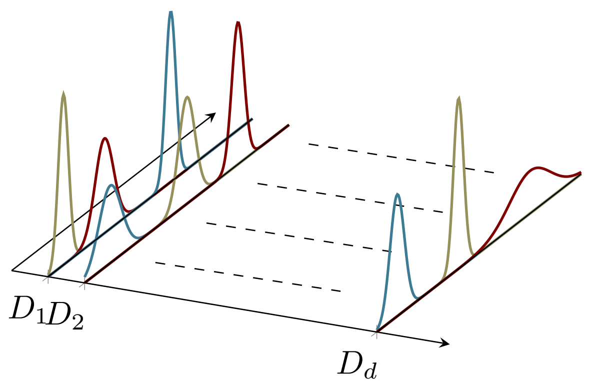


Figure 4.2: An example of the distribution of data points in each feature space given the proposed basic assumption.

More formally, we consider a UVG distribution as follows in which μ is the mean or center and σ is the standard deviation of the distribution,

$$N(f_e, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(f_e-\mu)^2}{\sigma^2}} \tag{4.4}$$

In each dimension, we have several UVG distributions with different μs and σs to create features in dimension e (e.g., f_e in D_e). We can summarize this representation with mixture of UVG distributions as follows,

$$p(f_e|\vec{\mu}, \vec{\sigma}) = \sum_{i\in[1,k]} \pi_i N(f_e, \mu_i, \sigma_i) \tag{4.5}$$

in which $\vec{\mu} = [\mu_1, \ldots, \mu_k]$, $\vec{\sigma} = [\sigma_1, \ldots, \sigma_k]$ and π_i is the weighting factor. It is expected that a good clustering data to have points with features that are selected from $p(f_e|\vec{\mu}, \vec{\sigma})$ with respective $N(f_e, \mu, \sigma)$s which are separated from each other. To see how this assumption affects the data we provide Figure4.2. The example shows a scenario where we have three different clusters and each $N(f_e, \mu, \sigma)$ has a different color. As we can see, in each feature space, we have a mixture of three UVG distributions with different μs,σs. There is also no synchronization between placements of μs across dimensions. In this case, the following proof is applicable that guarantees the meaningfulness of the distances in high dimensions.

Theorem 1 *Given the separation of points in each dimension, if coordinate system of Euclidean space is transformed from $\vec{o}_0$ to $\vec{o}_i$ where $\vec{o}_i \in c_i$ and points do not transform, it is possible to have meaningful distance values even for high dimensional data.*

Proof: $\forall \vec{p}_j \in c_i, \vec{p}_k \notin c_i$ different $N(f_e, \mu, \sigma)$s are dedicated from $p(f_e, x|\vec{\mu}, \vec{\sigma})$ to create $\vec{p}_j$s,$\vec{p}_k$s. Let's define a new norm $\|.\|_c$ which returns the distance of any point such as $\vec{p}_j$ based on the new coordinate system $\vec{o}_i$. We have,

$$\|\vec{p}_j\|_c = \textstyle\sum_{e=1}^{d} (\vec{p}_j.f_e - \vec{o}_i.f_e)^2 \tag{4.6}$$

Accordingly, $\|.\|_c^e$ returns the value of $\|.\|_c$ in dimension e. We know that when points are distributed with mixture of Gaussian distributions it means that μs and σs are in a way that $N(f_e, \mu, \sigma)$s are separated. For this reason, in each f_e, given $\vec{o}_i \in c_i$, we have, $\|\vec{p}_j\|_c^e < \|\vec{p}_k\|_c^e$. We have,

$$\|\vec{p}_j\|_c^e - \|\vec{p}_k\|_c^e = \varepsilon_e \tag{4.7}$$

in which ε_e is the difference of the distance between $\vec{p}_j, \vec{p}_k$ given center point $\vec{o}_i$ in dimension e. This result is the required effect that we want to make comparisons of any two points based on a center point $\vec{o}_i$ instead of $\vec{o}_0$. For all of the dimensions, we simply have the following outcome in which $Dist(\vec{p}_j, \vec{p}_k)$ is a new distance metric and equals to the sum of all ε_es.

$$Dist(\vec{p}_j, \vec{p}_k) = \left|\|\vec{p}_j\|_c - \|\vec{p}_k\|_c\right| = \left|\textstyle\sum_{e=1}^{d} \varepsilon_e\right| \tag{4.8}$$

As the number of dimensions increases the value of $Dist(\vec{p}_j, \vec{p}_k)$ also increases. The accretion is only true when $\vec{p}_j, \vec{p}_k$ are not in the same cluster. Otherwise, since they have the same UVG distribution, their overall distance is negligible.

Let's consider three points $\vec{p}_q, \vec{p}_j \in c_i, \vec{p}_e \notin c_i$, where $\vec{p}_q$ takes the role of query point, $\vec{p}_j$ has the nearest (D_{min}) and $\vec{p}_e$ has the farthest distance D_{max} to the $\vec{p}_q$. Since $\vec{p}_q, \vec{p}_j$ are in the same cluster, in each dimension they take values from the same UVG, therefore $Dist(\vec{p}_j, \vec{p}_q)$ is negligible. On the contrary, $\vec{p}_q, \vec{p}_e$ are not in the same cluster, so in each dimension they take values from different UVG distributions. This leads to the bigger values for $Dist(\vec{p}_j, \vec{p}_q)$

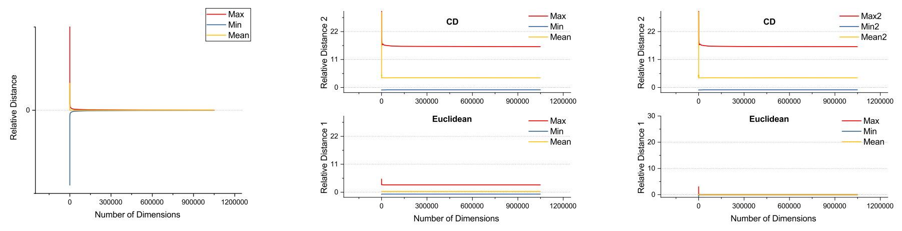
with more dimensions. Therefore, in (4.1), for the new distance metric, we have the following conclusion,

$$\frac{D_{max}-D_{min}}{D_{min}} \rightarrow D_{max} \tag{4.9}$$

In this result, when entropy is smaller for UVG distributions, the convergence gets closer to the D_{max} value. Regardless, the relative distance between any two points which are not in the same cluster converges to a non-zero value which guarantees the meaningfulness. ■

In Theorem 1, we proposed clustering norm ($\|.\|_c$) which guarantees meaningfulness of clustering. In $\|.\|_c$, the center of the clusters replace the origin vector in the coordinate system. Note that the square of the roots is not applied to (4.6). Consequently, the new formulation of the distance metric system further differentiates between points that are in the same cluster compared to the points that are not. For clustering of the points, first, we select k points randomly from set P. In each dimension, the K-Means algorithm is used separately to create clusters and their centers. In the end, we can simply classify those points that are closer to the $\vec{o}_i$ with the new distance metric. In this process, while it is possible to calculate distances between any two points with (4.8), the calculation of the $\|.\|_c$ is enough to make clusters. In this context, $\|\vec{p}_j\|_c$ corresponds to the $\|\vec{p}_j - \vec{o}_i\|_2$ the properties of the $\|.\|_2$ is inherited with the $\|.\|_c$.

4.1.4 Results



(a) 4 Clusters (b) 6 Clusters (c) 8 Clusters

Figure 4.3: Detailed analysis of meaningfulness in multivariate space in synthetic datasets.

In this section, we provide the results of our experiments for the proposed clustering distance. In this process, we use synthetic and non-synthetic datasets. The Table4.3 includes thirteen different datasets and provides associated features for them with, 1) number of the samples, 2) real number of the k, based on ground truth, 3) number of dimensions. The selections are popular datasets that can be retrieved from [189, 190] and cover a variety of properties.

An important argument for clustering is whether the clustering results are reliable. The reliability is a direct consequence of distance metrics for accurately differentiating between different points. A suitable measurement for calculating such reliability is (4.9) which measures the concentration effect. Let's first see how this phenomenon appears in multivariate data using Euclidean distances. To this end, we need data points which have consistent features from low dimensions to higher dimensions. Therefore, we use synthetic datasets in which members receive values similarly from small to a high number of features. An easy way to see the concentration effect is to generate points that obtain values in each dimension from an IID distribution within a specific scale (between 0 and 10000).

Table 4.1: The result table of distance analysis for non-synthetic datasets.

	Chiaretti	Chin	Christensen	Golub	Khan	Shipp	Su	Titanic	Wine	Iris	Colon	Diabetes	Mammographic
							Properties of Datasets						
Samples	128	118	217	72	63	77	102	1473	2201	105	60	752	830
Real K	6	2	3	2	4	2	4	3	3	3	2	2	2
Dimensions	10000	10000	1413	7129	2308	7129	5565	9	13	4	2000	8	5
							Results for Euclidean						
Mean	.000015	.0002	.0014	.0019	.0191	.0058	.0055	.353532491	.177041	.03486	.1425	.137712	4.140476
Min	-.025106	-0.08	-0.232	-0.17	-0.49	-.348	-0.32	-0.8891698	-0.8266	-0.529	-0.85	-0.9292	-0.80546
Max	.025753	.0906	.3032	.2134	.9980	.5352	0.478	8.02281545	4.76736	1.1258	5.666	13.1434	4.140477
							Results for Clustering Distance						
Mean	.119635	.0666	.1985	.1710	.1954	.3692	.0896	276593.731	110.520	7.5688	1.192	4.59559	41.24887
Min	-0.93509	-0.79	-0.64	-0.92	-0.89	-0.96	-0.96	-0.9999999	-0.9999	-0.998	-0.97	-0.9997	-0.99997
Max	14.39843	3.729	7.045	12.10	8.945	26.68	10.96	12183956.7	31221.7	591.84	31.75	3517.08	39929.28

In this way, we end up with a dataset which is generated randomly including data points that do not belong to any clusters. The feature spaces of the produced data points are from $\mathbb{R}^{2^1}$ to $\mathbb{R}^{2^{20}}$. This pattern provides a proper setting where we can track the convergence of relative distances. We directly use origin as a query point and calculate the corresponding distances with the Euclidean distance between any two pairs for 100 points. In this regard, the combination with Max value corresponds to (4.9) and Mean value corresponds to the overall differentiation between points. The Figure4.3a represents the results. As we can see, with an increase of the number of dimensions, Mean, Max and Min values in the graph quickly converge to 0 which is the expected result in concentration effect.

The underlying assumption of CD is the separation of points into k clusters in each dimension. Therefore, we should not use CD for calculation of distances in the previous dataset which generates points randomly without considering the separation of points. Instead, for the next experiment, we use a new dataset that produces data points based on a specific pattern. Accordingly, in each dimension, members are separated into four different clusters. The central aspect of this dataset is that points do not change their cluster memberships across dimensions. Based on this premise, members that form an assemblage in one dimension stay with the same crowds in all of the other dimensions. Also, the range of the values between clusters does not change. For instance, if c_i receives values between 10 and 20 in one dimension it receives contents from the same range in all of the dimensions. Based on this pattern, we consider the center of each cluster ($\vec{o}_i \in c_i$) to help us for calculation of the relative distances based on the proposed metric system.

The Figure4.3b represents the results. As we can see, this time, Euclidean distance does not converge to 0 even for dimensions of up to 2^{20}. This outcome shows the importance of correlations between points which can determine the meaningfulness of distance metrics. In this case, the features of each point are in a way that they retain almost the same distances to the query point for all of the features. The reason is that each point uses the same μ, σ for choosing its features. This design provides a clustering setting for Euclidean distance metric similar to what we accomplish with CD metric system. Therefore, the distances remain meaningful. To measure the CD, among four clusters we randomly select center of a cluster as query point. The selection helps us to see that changing the coordinate system from origin to $\vec{o}_i \in c_i$ would mitigate the concentration effect in practice. As we can see in this illustration, CD also does not converge to the value of 0. However, note that for both Mean and Max results, CD converges to a higher value compared to the Euclidean distance. The reason is that unlike the $\vec{o}_0$ which had the role of a query point for Euclidean metric system, the center of the cluster for CD traces the changes of the clustering points much better.

In the previous dataset, the pattern that we have designed for the creation of points is restrictive on the range of values across dimensions. Consequently, points of the same cluster would receive contents based on the same μs, σs for all of their features. In this way, we helped Euclidean distance for not falling into the trap of concentration effect. In the next dataset, we use the same design of the previous dataset but remove the restriction of having the same μs across dimensions. Subsequently, points of the same cluster still receive their contents from the same

μ, σ. However, the values of μs may be different in each dimension. The Figure4.3c shows the results. This time, the outcomes for Euclidean distance is similar to the effects of randomly generated data and quickly converges to 0. On the other hand, CD, same as the previous scenario does not converge to the value of 0. These results indicate that even in simple situations where points are decomposable in each dimension, the Euclidean distance can quickly fall for the concentration effect. This outcome is valid unless we use the query points that can trace the change of clusters in each dimension. In the proposed distance metric careful attention is given to calculate the distances based on these tracings which prevent the concentration effect.

In addition to the synthetic datasets, in Table4.3 we have non-synthetic datasets which respectively produce less meaningful distance values compared to the proposed metric system. Similar to the Figure4.3c the results are for maximum, minimum and mean values. The results confirm the experiments on synthetic datasets. More specifically, for the Max values, we had a minimum of 5.6 times higher differentiation for Colon dataset and a maximum of 1518663.46 times higher differentiation for Titanic dataset. Respectively, for the mean values, we had a minimum of 9.96 times higher differentiation for the Mammographic dataset and a maximum of 782371.46 times higher differentiation for Titanic dataset. According to our observations, in general, the results depend on the selection of the query point, whether points can differentiate themselves based on the reference and if points follow the underlying assumptions of the proposed methods.

4.2 Condensed Silhouette

4.2.1 Introduction

A fundamental requirement for the K-Means based clustering algorithms is to select initial seeds from clustering points. Current methods of seed initialization [191, 192] either rely on random selections or prespecified patterns such as K-Means++ [193]. Both of these strategies do not use any information from existing data, which can lead to inaccuracies in the clustering results. To solve this problem, we can use a filtering process that chooses the clustering result with the best score among several trials. For this purpose, we need clustering scores, which can be achieved with clustering evaluation metrics.

It has been shown that different evaluation methods can be more effective on certain problems [194, 195]. In general, one of the most effective evaluation metrics is the Silhouette algorithm [196, 197, 198]. An extensive study in [199] shows that the Silhouette index obtains the best results in many cases. Another study found out that Silhouette has the most correlation with human evaluations [200]. Other studies [201, 202] further confirm the effectiveness of this method. However, one drawback of using this method is the resources required to calculate Silhouette score. In contrast, the within-cluster sum of squares (WCSS) integrates with K-Means clustering perfectly and does not need extra resources. This is probably an important factor that led to the popularity of this method and its usage as the default method for current clustering libraries such as sickit-learn [203].

In this section, we propose Condensed Silhouette as an optimized clustering evaluation method for K-Means. The goal is to replace essential elements of Silhouette algorithm with relevant elements of the K-Mean algorithm to reduce the required resources for calculation of the Silhouette value. For clustering, we filter out the results which do not preserve compactness based on Multivariate Gaussian Distribution for its clusters. We also consider the distances of the clusters from each-other. In this way, next to compactness, we also encourage the results where clusters have higher minimum distances. The results show the effectiveness of our method. For accuracies, we could provide close results compared to the original Silhouette. In the 14 datasets, the average of the accuracy differences between Condensed Silhouette and Silhouette is only 0.0042. For computational resources, our benchmark is WCSS, which does not need additional computations for calculating its evaluation score. In the 14 datasets, Condensed Silhouette needed 5.4s, and Silhouette needed 272.73s more time compared to the WCSS. This means that Condensed Silhouette is 50.5 folds closer to the benchmark compared to the Silhouette.

4.2.2 General Model

To make k clusters out of members of P, agents need a clustering algorithm. In clustering algorithms, based on different initialization settings, the membership of points to clusters can change dramatically. The most important initialization setting, which is common in K-Means based algorithms, is choosing initial seeds. We take the results of K-Means based clustering algorithms with different initial seeds as $\mathrm{C}_1, \ldots, \mathrm{C}_t$ where t is the maximum number of trials. In this regard, each C_i represents an instance of a final clustering result. We have $\mathrm{C}_i = \{c_1, \ldots, c_k\}$ which are created from initial seeds $\vec{p_1}, \ldots, \vec{p_k}$. In this notation, each $\vec{p_j}$ is an initial seed that is taken from set P. Respectively, the centers of the clusters are denoted by $\vec{o_1}, \ldots, \vec{o_k}$. In this notation, each vector $\vec{o_i}$ is the center of cluster c_i. If there is a score for each C_i, the agent can choose the best C_i.

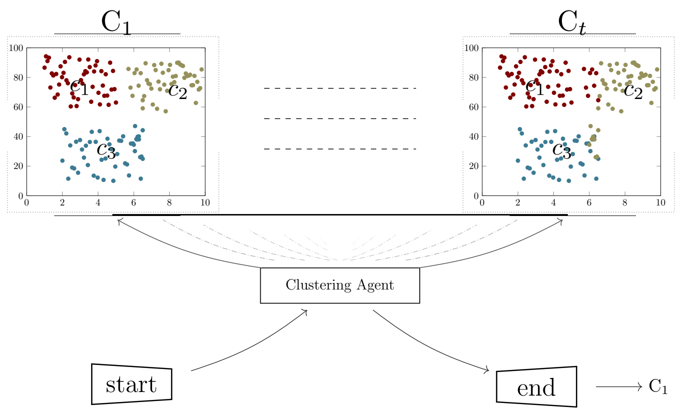


Figure 4.4: The overall representation of the filtering process in clustering.

For clarification of the notifications and the filtering process, see Figure 4.4. In this example, we have $\{C_1, \ldots, C_t\}$ trials. Note that each C_i contains three clusters, $\mathrm{C}_i = \{c_1, c_2, c_3\}$) which means $k = 3$. The filtering process starts with an agent, which triggers the clustering algorithm t times. The clustering algorithm provides the agent with different results by conducting a clustering process using t different sets of initial seeds. The agent uses an evaluation algorithm such as Silhouette and chooses the result with the best evaluation outcome (in this example, C_1). In this model, our goal is to propose an evaluation metric that is similar to the Silhouette algorithm and also integrates with the K-Means algorithm to reduce the required resources.

4.2.3 The Proposed Method

K-Means is an iterative clustering algorithm that refines the assigned clusters through many iterations. Initially, the desired number of clusters is selected, which is equal to the number of centroids. Every data point is allocated to a cluster based on its distances to the centroids of all the clusters. More specifically, for the task of clustering, the K-Means algorithm follows two steps until convergence. First, for each $\vec{o_i} \in c_i$, members of P which are closer to the $\vec{o_i}$ become the new members of c_i. Second, for each c_i the position of its corresponding $\vec{o_i}$ changes as follows,

$$\vec{o_i} = \tfrac{1}{|c_i|} \textstyle\sum_{\vec{p}_j \in c_i} \vec{p}_j \tag{4.10}$$

in which $|c_i|$ is the number of points in c_i. The 'means' in the K-Means is averaging of the points ($\sum_{\vec{p}_j \in c_i} \vec{p}_j$), to the number of the points in the cluster ($|c_i|$). After convergence, we satisfy the following objective function, which means distances of points to their cluster center are less than or equal to centers of other clusters.

$$\forall \vec{p}_j, \vec{o_i} \in c_i, \vec{o_h} \notin c_i, \|\vec{p}_j - \vec{o_i}\|_2 \leq \|\vec{p}_j - \vec{o_h}\|_2 \tag{4.11}$$

This outcome is related to the main result of the iterative process in K-Means which makes sure that data points ($\vec{p}_j$s) are closer to the centroids of their respective clusters ($\vec{o}_i s \in c_i$) compared to the centroids of the clusters that they do not belong ($\vec{o}_i s \notin c_i$). The objective function (2) is the most important quality consideration of clustering in K-Means [204]. For the filtering, we repeat this algorithm t times with different sets of initial seeds. The main concern in the filtering process is to help agents for choosing the best C_i, which requires an overall score for each C_i.

4.2.4 Silhouette Evaluation

To determine the quality of each C_i, an evaluation degree has to evaluate the elements of C_i. For this purpose, the Silhouette algorithm focuses on two aspects of the elements of C_i. The first criterion is the compactness of c_is. For clustering, it is desired to have clusters with smaller entropies. To measure the compactness in Silhouette algorithm, we have,

$$a(i) = \tfrac{1}{|c_h|-1} \textstyle\sum_{\vec{p}_i, \vec{p}_j \in c_h, i \neq j} \|\vec{p}_i - \vec{p}_j\|_2 \tag{4.12}$$

As we can see, a(i) is the mean distance between point $\vec{p_i}$ and all of the points in the same cluster. Therefore, in this method, the distances between every two points that are in the same cluster are measured.

For second criterion, next to the compactness, an important observation for measurement of the clusters is the relationship of each cluster member of C$_i$ with other members of C$_i$. This means that it is desired to have clusters that are as far from each other as possible. To calculate this criterion, we have,

$$b(i) = \min \tfrac{1}{|c_h|} \textstyle\sum_{\vec{p_i} \in c_h, \vec{p_j} \neq c_h} \|\vec{p_i} - \vec{p_j}\|_2 \tag{4.13}$$

In this regard, the $b(i)$ is the minimum value of the mean results of $\vec{p_i}$ compared to all of the points that are not in the same cluster as $\vec{p_i}$. Therefore, for each point $\vec{p_i}$, we have to calculate the mean of distances of $\vec{p_i}$ with all of the members of other clusters. The cluster that has the minimum value is chosen for b(i). For each point $\vec{p_i}$ we have to get its respective a(i), b(i) and combine them as follows,

$$s(i) = \begin{cases} \frac{b(i)-a(i)}{max\{a(i),b(i)\}} & C_i > 1 \\ 0 & C_i = 1 \end{cases} \tag{4.14}$$

We can rewrite this result as follows, which clearly shows that $s(i)$ can take a value between -1 and 1 ($-1 \leq s(i) \leq 1$). For clustering score, it is desired that $s(i)$ gets closer to 1.

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)} & a(i) < b(i) \\ 0 & a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1 & a(i) > b(i) \end{cases} \tag{4.15}$$

For the final value of Silhouette, the mean of all of the values of $s(i)$s is used. In the next section, we provide an alternative approach for calculating $a(i), b(i)$, which can significantly reduce the computation of the Silhouette algorithm.

4.2.5 Condensed Silhouette

In the Condensed Silhouette we want to stay true to the methodologies of Silhouette. The main difference is to replace the evaluation of $a(i)$ and $b(i)$ with alternatives that require less computational resources. To evaluate $a(i)'$ which is the alternative approach for compactness of clusters, instead of the current methodology in Silhouette, we use the minimization of the within-cluster sum of squares (WCSS). This is a simple and yet effective method that can be written as follows,

$$a(i)' = WCSS = \sum_{c_i \in C} \sum_{\vec{p}_j \in c_i} \|\vec{p}_j - \vec{o}_i\|_2, \vec{o}_i \in c_i \tag{4.16}$$

As we can see, in this method, stationary points (centers) and the distances with their members are used as a reference to measure the compactness of clusters. This is in contrast to the Silhouette method, where the distances of all pairs have to be measured. If shapes of the clusters are assumed to be Multivariate Gaussian -otherwise K-Means should not have been used- then WCSS is a good measurement which makes the current methodology of measuring a(i) in Silhouette redundant.

To calculate the number of required distances for a(i) in Silhouette, we can use the combinations formula, as follows,

$$\begin{pmatrix} n \\ j \end{pmatrix} = \frac{n!}{j!(n-j)!} \tag{4.17}$$

which provides the number of possible combinations of j objects from a set of n objects. In this case, we want, number of the unique pairs. Therefore, j is 2 which gives us following equation,

$$dists = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} \tag{4.18}$$

For instance, with a result of 3 clusters, and 100 points, in each cluster, we have $4950 \times 3 = 14850$ distance calculations between pairs of point. Comparatively, the only overhead of $a(i)'$, is $\sum_{\vec{o}_i \in C}$ part from (7) which in this case is adding three numbers. Note that the calculations of $\sum_{\vec{p}_j \in c_i} \|\vec{p}_j - \vec{o}_i\|_2$ is already included in the convergence of the K-Means algorithm and does not have any overhead for calculating $a(i)'$. Therefore, measuring the compactness of clusters for $a(i)'$ is basically free when it is used in combination with K-Means algorithm.

The evaluation of $b(i)$ in Silhouette which measures the quality of distances between clusters, needs distances between every two nodes which are not in the same clusters. If we combine this requirement with evaluation of a(i), the (8) has to be used for all points in all of the clusters. In this regard, with 3 clusters and 100 points in each cluster, we have $\frac{400\times(400-1)}{2} = 79,800$ number of measurements of the distances.

Let us define $b(i)'$ as an alternative for $b(i)$. For the evaluation of $b(i)'$, we only compare the centers with each other. This is a reasonable choice because the center of a cluster is created from features of all of the points that are in the same cluster. Therefore, centers are good representations of overall points in the same clusters.

$$b(i)' = \min_{i\neq j} \left\|\vec{o}_i - \vec{o}_j\right\|_2, \vec{o}_i \in c_i, \vec{o}_j \in c_j \tag{4.19}$$

In this way, we make separate calculations for every pair of points redundant. Instead, we only calculate distances between every pair of centers. As a result, the number of distances for the proposed method is reduced to,

$$dists = \frac{k(k-1)}{2} \tag{4.20}$$

Following the previous example, with a result of 3 clusters, and 100 points in each cluster, we have $\frac{3\times(3-1)}{2} = 3$ number of distances. To calculate the final score ($s(i)'$), we use (5),(6) from Silhouette algorithm, as follows,

$$s(i)' = \begin{cases} 1 - \frac{a(i)'}{b(i)'} & a(i)' < b(i)' \\ 0 & a(i)' = b(i)' \\ \frac{b(i)'}{a(i)'} - 1 & a(i)' > b(i)' \end{cases} \tag{4.21}$$

For the final value of Silhouette, the mean of all of the values of $s(i)'$s is used. Similarly, $s(i)'$ can take a value between -1 and 1 ($-1 \leq s(i)' \leq -1$). For clustering, higher value of $s(i)'$ is desired.

4.2.6 Results

In this section, we provide the analysis for computation and accuracy results. For accuracy, we expect Condensed Silhouette to have similar outcomes compared to the Silhouette. In the computational analysis, we expect resource consumption to be closer to the WCSS. We are specifically interested in mitigating the growth for resource consumption of the Silhouette when the number of the points increases. For all the experiments, we use the standard implementations in sickit-learn library. Accordingly, for implementing Condensed Silhouette we forked the related Silhouette codes from sickit-learn, and changed the codes where it was needed.

4.2.7 Accuracy Analysis

The results of the experiments in this section are mean percentages of the correct labeling for clustering compared to the ground truth over 100 trials. Such comparison is an assignment problem where we use the algorithm introduced in [205]. In each trial, we take ten samples using K-Means algorithm. Next, an evaluation metric is applied to all ten samples. In the end, the sample which provides the best value based on the evaluation metric is chosen for that trial. For a given dataset, for each trial, the number k of clusters was set to the number of ”classes” known to exist in the dataset. The numbers in the tables are the mean values for over 100 trials. For the clustering result, we compare our method with several other approaches. The first approach is WCSS, which is the default method that is used in sickit-learn library. Next to the WCSS, we also compare our method with Silhouette value and Davies Bouldin score. Both of them are prominent evaluation methods that have proper implementations in sickit-learn.

Table 4.2: The result table for accuracy analysis of seven real datasets, using gene expression microarrays.

	alon	chiaretti	golub	gravier	singh	tian	christensen
Samples	62	128	72	168	102	173	217
Real K	2	6	2	2	2	2	3
Features	2000	10000	7129	2905	10000	10000	1413
Accuracy							
WCSS	53.22	36.40	57.73	52.17	57.86	56.86	99.53
DaviesBouldin	53.19	41.4	64.5	65.87	57.49	57.8	93.35
Silhouette	50.37	39.66	66.13	66.54	57.75	57.79	99.53
CondSilhouette	50.40	40.14	64.79	66.34	57.77	57.80	99.53

For the experiments, at first, we use gene expression microarray datasets, which have a high number of features [189, 190]. In these datasets, number of the samples are relatively small. For the accuracy results, see Table 4.2. The upper part of the table provides associated features for datasets with, 1) number of the samples, 2) real number of the k, based on ground truth, 3) number of the features. Respectively, the lower part of the table provides the mean values for 100 iterations of clustering results on each dataset. Comparing WCSS with Condensed Silhouette, on four datasets, Condensed Silhouette is better with at most 14.17% higher accuracy for Gravier and at least 0.94% higher accuracy for Tian. For datasets Alon and Singh, the WCSS provides 2.82% and 0.09% better accuracies compared to the Silhouette. In Christensen, both methods had the same results. Overall, Condensed Silhouette was a better method with a cumulative higher accuracy of 23.0% in all seven datasets.

If we compare Davies Bouldin score with Condensed Silhouette, on four datasets, the accuracy of Condensed Silhouette is higher. The most increase of the accuracy is for Christensen with 6.15%, and the least increase of the accuracy is for Singh with 0.28% higher accuracy. On the other hand, Davies Bouldin score could provide higher accuracy on two datasets with at most 2.79% higher accuracy on Alon and at least 1.26% higher accuracy on Chiaretti. Overall, Condensed Silhouette was a better method with a cumulative higher accuracy of 3.17% in all seven datasets. For the Silhouette, the results are very close. On four datasets, Condensed Silhouette was better than Silhouette with at most, 0.48% better accuracy for Chiaretti and at least 0.01% better accuracy for Tian. For the Golub and Gravier dataset, Silhouette had better accuracy with 1.34% and 0.2% compared to the Condensed Silhouette. For Christensen both methods had the same accuracy. On the cumulative results, Silhouette could slightly edge Condensed Silhouette by one percent.

Table 4.3: The result table for accuracy analysis of seven real datasets

	cmc	IRIS	unbalanced	wdbc	bupa	contraceptive	mammographic
Samples	1473	150	856	569	345	1473	830
Real K	3	3	2	2	2	3	2
Features	9	4	32	30	6	9	5
Accuracy							
WCSS	40.01	89.32	56.30	85.41	55.35	39.95	68.55
DaviesBouldin	40.06	89.33	56.30	85.41	55.25	40.05	67.88
Silhouette	40.11	89.32	56.30	85.41	55.23	40.11	67.53
CondSilhouette	40.11	89.33	56.30	85.41	55.26	40.11	68.55

In Table 4.3, we use seven more real datasets. Unlike the gene expression microarray datasets, in these datasets, the number of the features compared to number of the points is relatively small. For WCSS and Condensed Silhouette, on three datasets Condensed Silhouette had better results, with at most 0.1 better accuracy for cmc. Respectively, on one dataset, WCSS was better than Condensed Silhouette, with 0.09% better accuracy for Bupa. On the cumulative results, the Condensed Silhouette could slightly outperform WCSS, with 0.18% better accuracy.

Between Davies Bouldin and Condensed Silhouette, the Condensed Silhouette had better results in four datasets with at most 0.67% higher accuracy in Mammographic. In three other datasets, the outcomes for both methods are the same. For the cumulative results, the Silhouette is better with 0.79% higher accuracy. When comparing Condensed Silhouette to Silhouette, the accuracy of Condensed Silhouette was better on three datasets with up to 1.02 higher accuracy and down to 0.01 higher accuracy. On four datasets, two methods had the same accuracy. On the cumulative results, the Silhouette could slightly outperform Condensed Silhouette, with 1.06 better accuracy.

4.2.8 Computational Analysis

To analyze the computational requirements of the methods we use time.process_time() in Python, which is specifically designed to calculate the time it takes to execute a process without considering time elapsed during sleep. To run the experiments, we used AMD 2700x CPUs on idle Windows 10 machines with single cores. Similar to previous experiments, the results are attained over 100 trials. Respectively, the numbers in the tables are the times to do 100 trials in seconds (s). As mentioned before, the best process time is for the WCSS method, and the goal is to get as close to this method as possible. Table 4.4, provides the computational time for four methods over seven gene expression microarray datasets. The computational time for Davies Bouldin score is, on average, 9.57s higher than WCSS. The maximum computational difference is for Tian, with 21.53s higher computation than WCSS. The minimum computational difference is for Alon with 1.92s higher computation than WCSS.

The average computational time for Silhouette is 10.06s higher than WCSS. The highest and lowest computational times are for Tian and Alon with 25.7s and 1.71s higher benchmark times than WCSS. Comparatively, the Condensed Silhouette, on average, has only 0.37s higher benchmark times than WCSS. An important note here is that the computational time for Chiaretti is lower for the Condensed Silhouette. This result is possible because the convergence time on K-Means for trials of Condensed Silhouette is lower than the convergence time

Table 4.4: The result table for computational analysis of seven real datasets, using gene expression microarrays.

	alon	chiaretti	golub	gravier	singh	tian	christensen
Samples	62	128	72	168	102	173	217
Real K	2	6	2	2	2	2	3
Features	2000	10000	7129	2905	10000	10000	1413
Computation							
WCSS	5.18	100.78	24.50	23.07	44.92	97.78	12.43
DaviesBouldin	7.10	106.62	31.95	31.78	62.34	119.31	16.60
Silhouette	6.89	109.15	29.82	30.01	60.28	123.48	19.45
CondSilhouette	5.56	94.64	24.79	23.68	49.64	98.84	14.12

on K-Means for trials of WCSS. However, this is a rare instance, and in general, WCSS should provide better performance. Overall, relative to the base benchmark of the WCSS, our method provides 26.98 folds closer benchmark time than Silhouette.

Table 4.5: The result table for computational analysis of seven real datasets

	cmc	IRIS	unbalanced	wdbc	bupa	contraceptive	mammographic
Samples	1473	150	856	569	345	1473	830
Real K	3	3	2	2	2	3	2
Features	9	4	32	30	6	9	5
Computation							
WCSS	3.73	1.76	3.14	2.21	1.73	3.70	1.81
DaviesBouldin	5.42	2.82	4.78	3.37	3.17	5.51	3.14
Silhouette	75.29	3.00	26.18	11.92	5.79	75.39	22.82
CondSilhouette	4.29	1.92	3.40	2.29	2.46	4.25	2.26

In table Table 4.5, we provide the computational analysis for seven additional datasets. The DaviesBouldin score, on average, needed 1.44s more time than WCSS to complete its tasks. The lowest time difference is for IRIS with 1.06s more computational time. The highest time difference is for Contraceptive with 1.81s more than computational time. For the Silhouette, on average, the time increase compared to the WCSS is 28.90s. The time increase could go up to 75.39s for Contraceptive, which has the highest number of the points. Comparatively, on average, the increase of the time for the Condensed Silhouette is only 0.39s. Overall, for the average values, relative to the base benchmark of the WCSS, our method was 74.10 times closer to the benchmark than Silhouette.

4.3 Subspace Clustering with Data Augmentation

4.3.1 Introduction

Recent advances in technology have provided massive amounts of complex high dimensional data for computer vision and machine learning applications [206, 207, 208]. High dimensionality has adverse effects, including confusion of algorithms with irrelevant dimensions and *curse of dimensionality* as well as increased computation time and memory. This motives us to explore techniques for representing high-dimensional data in lower dimensions. In many practical applications such as face images under various illumination conditions [209] and handwritten digits [210], high-dimensional data can be represented by union of low-dimensional subspaces. The subspace clustering problem aims at finding these subspaces [211, 212, 167]. In particular, the objective of subspace clustering is to find the number of subspaces, their basis and dimensions, and assign data to these subspaces [213].

Conventional subspace clustering algorithms assume that data lie in linear subspaces [214, 215, 216, 217]. In practice, however, many datasets are better modeled by non-linear manifolds. To deal with this issue, deep subspace clustering (DSC) methods [218, 219, 220, 221, 222, 223] have been proposed which essentially which learn the unsupervised nonlinear mappings by projecting data into a latent space. Deep subspace clustering networks have shown promising performances on various datasets.

Deep learning techniques are prone to overfitting. Data augmentation is often presented as a type of regularization to mitigate this issue [224, 225]. While data augmentation for deep learning-based methods have proven to be beneficial, the current framework of DSC networks is unable to take the full advantage of data augmentation. In this work, we modify the DSC framework and propose a model that can incorporate data augmentation into DSC.

An important difference between data augmentation in subspace clustering and data augmentation in supervised tasks is the fact that as apposed to supervised tasks, we do not have ground-truth labels for the existing samples in the subspace clustering algorithms. Corresponding to the fact that objects remain the same even if we slightly transform them, in supervised deep learning models, transformations of an existing sample are trained to be predicted with a consistent label similar to the ground-truth label of the original sample. How can one convey such property in an unsupervised subspace clustering task, where the data does not have the ground-truth labels?

A DSC model should favor functions that give consistent outputs for similar data points with a slight difference in their percept. To achieve this, we optimize a consistency loss that is based on temporal ensembling. We input plausible transformations of existing samples into the model and require the autoencoders of the model to map the transformations to consistent subspaces similar to the subspace of the original data.

Efficient augmentation policies improve the performance of the deep networks. However, not all the image transformations construct efficient augmentation policies. Efficient augmentation policies can be different from a dataset to another [113]. In supervised applications, the validation set is often used to manually search among transformations such as rotation, horizontal flip, or translation by a few pixels to find efficient augmentations. Manual augmentation needs prior knowledge and expertise, and it can only search among a handful of pre-defined trials. Some methods automate this search for classification networks [113]. However, these methods are only designed for the classification task and cannot be applied to the task of subspace clustering. This is because we do not have a validation or training set in subspace clustering. We overcome this issue by providing a simple yet effective method for finding efficient augmentation policies using a greedy search and use mean Silhouette scores to evaluate the effect of different augmentation policies on the performance of our proposed model.

4.3.2 Related Work

Clustering Methods With Augmentation. A recent method proposes a technique for deep embedded clustering algorithms with augmentations [226, 227]. In the pre-training stage they use augmentations in training autoencoders, and in the fine-tuning stage they encourage the augmented data to have the same centroid as their corresponding data. To the best of our knowledge, we are the first to propose an augmentation framework for deep subspace clustering algorithms.

Self-expressiveness Models in Subspace Clustering. Let $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{D\times N}$ be a collection of N signals $\{\mathbf{x}_i \in \mathbb{R}^D\}_{i=1}^N$ drawn from a union of n linear subspaces $\mathcal{S}_1 \cup \mathcal{S}_2 \cup$

$\cdots \cup \mathcal{S}_n$. Given $\mathbf{X}$, the task of subspace clustering is to find sub-matrices $\mathbf{X}_\ell \in \mathbb{R}^{D\times N_\ell}$ that lie in $\mathcal{S}_\ell$ with $N_1 + N_2 + \cdots + N_n = N$.

Due to their simplicity, theoretical correctness, and empirical success, subspace clustering methods that are based on *self-expressiveness property* are very popular [228]. Self-expressiveness property can be stated as

$$\mathbf{X} = \mathbf{X}\mathbf{C} \;\; s.t \;\; \text{diag}(\mathbf{C}) = \mathbf{0}, \tag{4.22}$$

where $\mathbf{C} \in \mathbb{R}^{N\times N}$ is the coefficient matrix. There may exist many coefficient matrices that satisfy the condition in (4.22). Among those, *subspace preserving* solutions are especially of interest to self-expressiveness based subspace clustering methods. Subspace preserving property states that if an element in $\mathbf{C}$ is non-zero, the two data points in $\mathbf{X}$ that correspond to this coefficient are in the same subspace.

Self-expressiveness based methods combine these two properties and solve a problem of the form:

$$\min_{\mathbf{C}} \mathcal{L}_{\text{S.E.}}(\mathbf{C}, \mathbf{X}) + \lambda_1 \mathcal{L}_{\text{S.P.}}(\mathbf{C}), \tag{4.23}$$

where λ_1 is a regularization constant, $\mathcal{L}_{\text{S.E.}}$ and $\mathcal{L}_{\text{S.P.}}$ impose the self-expressiveness and subspace-preserving properties, respectively. Most of the linear methods use $\mathcal{L}_{\text{S.E.}}(\mathbf{C}, \mathbf{X}) = \|\mathbf{X} - \mathbf{X}\mathbf{C}\|_F^2$. However, for $\mathcal{L}_{\text{S.P.}}(\mathbf{C})$, different methods use various regularizations, including ℓ_1-norm, ℓ_2-norm and nuclear norm [214, 228, 229].

In recent years, deep neural network-based extensions were introduced to self-expressiveness based models [218, 219, 220, 221]. For these methods, x_is do not need to be drawn from a union of linear subspaces. Instead, they use autoencoder networks to map the data points to a latent space where data points lie into a union of linear subspaces and exploit the self-expressiveness and subspace-preserving properties in the latent space. Let $\mathbf{Z} \in \mathbb{R}^{d\times N}$ be the latent space features developed by the encoder in the autoencoders. Deep subspace clustering networks solve a problem of the form:

$$\min_{\mathbf{\Theta}} \mathcal{L}_{\text{S.E.}}(\mathbf{C}, \mathbf{Z}) + \lambda_1 \mathcal{L}_{\text{S.P.}}(\mathbf{C}) + \lambda_2 \mathcal{L}_{\text{Rec.}}(\mathbf{X}, \hat{\mathbf{X}}), \tag{4.24}$$

where λ_1 and λ_2 are regularization constants, Θ is the union of trainable parameters, $\hat{\mathbf{X}}$ is the reconstruction of $\mathbf{X}$ and the output of the decoder, and $\mathcal{L}_{\text{Rec.}}(\mathbf{X}, \hat{\mathbf{X}}) = \|\mathbf{X} - \hat{\mathbf{X}}\|_F^2$ is the reconstruction loss in training the autoencoder. Once a proper $\mathbf{C}$ is found from (4.23) or (4.24),

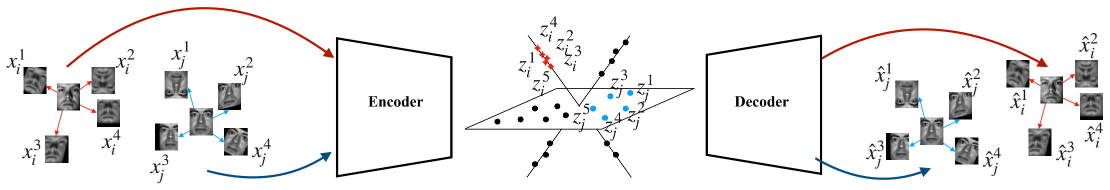


Figure 4.5: An overview of the proposed deep subspace clustering networks with data augmentation. The existing data points x_i and x_j are transformed into x_i^t and x_j^t in each iteration by an augmentation policy. However, the autoencoder learns to keep their latent space features within consistent subspaces.

spectral clustering methods [230] are applied to the affinity matrix $\mathbf{W} = |\mathbf{C}| + |\mathbf{C}|^T$ to obtain the segmentation of the data $\mathbf{X}$.

4.3.3 Deep Subspace Clustering Networks with Data Augmentation

The human brain considers an object to remain the same, even if the percept changes slightly. Correspondingly, when data augmentation is used in supervised deep learning models, transformations of existing samples are trained to predict consistent labels similar to the ground-truth label of original samples. Conveying the same insight, we argue that a DSC model should favor functions that give consistent outputs for similar data points. We approach this property by keeping the estimated subspace membership of data points consistent when an augmentation policy is applied to them. During the training process, we smooth the predictions for the subspace memberships via temporal ensembling of estimated affinity matrices from previous iterations.

Let $\mathbf{X}^t = [\mathbf{x}_1^t, \cdots, \mathbf{x}_N^t] \in \mathbb{R}^{D \times N}$ be the transformed version of N existing data points $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$ at the iteration t. $\mathbf{X}^t$ is the observation at time t when an augmentation policy is applied to the existing data points $\mathbf{X}$.

Our model can be applied to a variety of DSC networks. In this section, we consider a general form that consists of an encoder that takes $\mathbf{X}^t$ as an input and generates latent space features $\mathbf{Z}^t$. The latent space features are reconstructed by a self-expressive layer with parameters $\mathbf{C}^t$. That is, $\mathbf{Z}^t\mathbf{C}^t$ is fed to the decoder to develop $\hat{\mathbf{X}}^t$, which is a reconstruction of $\mathbf{X}^t$. Figure 4.5 shows an overview of this model. Note that such a model includes a fully-connected layer that connects all the samples in the mini-batch (the self-expressive layer). Thus, the number of data points and their orders cannot be changed during the training. We keep a placeholder with N

fields that correspond to the existing samples and feed $\mathbf{X}^t$ to this placeholder at every training step t. The permutation of samples in $\mathbf{X}^t$ remains the same.

As mentioned, we aim for an autoencoder that preserves the subspace membership of slightly transformed inputs. Let $\mathbf{C}^t$ be the coefficient matrix that is constructed at the t-th iteration of a subspace clustering algorithm. In addition, let $\hat{\mathbf{Q}}$ be an existing estimation of subspace membership matrix, whose rows are one-hot vectors denoting the subspace memberships assigned to different samples. The multiplication of $\hat{\mathbf{Q}}^T$ and $|\mathbf{C}^t|$ gives a matrix whose (i, j)th element shows the contribution of the samples assigned to the i-th subspace in reconstructing the j-th sample. For a perfect subspace-preserving coefficient matrix, $\hat{\mathbf{Q}}^T|\mathbf{C}^t|$ has only one non-zero element in each row.

For each sample j, the maximum value in the j-th row of $\hat{\mathbf{Q}}^T|\mathbf{C}^t|$ can point to a new estimate for its subspace membership. Therefore, a prediction of subspace membership matrix at the iteration t can be calculated as follows

$$\mathbf{Q}^t = \text{Softmax}(\hat{\mathbf{Q}}^T|\mathbf{C}^t|), \tag{4.25}$$

where Softmax$(\cdot)$ corresponds to the softmax function on the rows of its input. We refer to $\mathbf{Q}^t$ as *temporal* subspace membership matrix.

The temporal subspace membership matrix $\mathbf{Q}^t$ estimates the subspace memberships for the current observation $\mathbf{X}^t$. Note that because of the randomly augmented inputs, the coefficient matrix $\mathbf{C}^t$ can undergo sudden changes in different time frames. While it is fine to have different coefficient matrices for slight transformations of data, we are interested in maintaining persistent subspace membership matrices $\mathbf{Q}^t$. Thus, we propose a subspace membership consistency loss.

We keep an exponential moving average (EMA) of $\mathbf{C}^t$s, the coefficient matrices, to provide a smooth temporal ensemble for the coefficient matrix. Thus, in addition to the *temporal* subspace membership matrix in (4.25), in each training iteration, we can calculate another membership matrix corresponding to the temporal ensemble of coefficient matrices in prior iterations. We refer to this membership matrix as $\mathbf{Q}^t_{\text{Ens.}}$.

Let $\mathbf{C}^{t-1}_{\text{EMA}}$ be the EMA of coefficient matrices until the iteration $t-1$, and $\mathbf{C}^t$ be the calculated update for the coefficient matrix at the iteration t. The EMA of the coefficient matrix at the iteration t can be updated as follows

$$\mathbf{C}^t_{\text{EMA}} = \alpha \mathbf{C}^{t-1}_{\text{EMA}} + (1-\alpha)\mathbf{C}^t, \tag{4.26}$$

where $0 < \alpha < 1$ is the smoothing factor. Using $\mathbf{C}^t_{\text{EMA}}$ we can calculate $\mathbf{Q}^t_{\text{Ens.}}$ as

$$\mathbf{Q}^t_{\text{Ens.}} = \text{Softmax}(\hat{\mathbf{Q}}^T |\mathbf{C}^t_{\text{EMA}}|), \tag{4.27}$$

where $\hat{\mathbf{Q}}$ is the same prior membership matrix as in (4.25).

Note that $\mathbf{Q}^t_{\text{Ens.}}$ provides more consistent subspace membership predictions as compared to $\mathbf{Q}^t$. To encourage the autoencoders to favor functions that preserve the subspace memberships even for differently transformed observations $\mathbf{X}^t$, we propose the subspace membership consistency loss as follows:

$$\mathcal{L}_{\text{Cons.}}(\mathbf{Q}^t_{\text{Ens.}}, \mathbf{Q}^t) = \text{CE}(\mathbf{Q}^t_{\text{Ens.}}, \mathbf{Q}^t), \tag{4.28}$$

where CE$(\cdot)$ denotes the cross-entropy function. $\mathcal{L}_{\text{Cons.}}$ penalizes the temporal changes to the subspace memberships if they are inconsistent with the temporal ensemble of subspace memberships $\mathbf{Q}^t_{\text{Ens.}}$.

Full Objective. We train the networks iteratively with two steps of subspace clustering and subspace membership consistency in each iteration. In the subspace clustering step, the loss function of the subspace clustering algorithm of choice (4.24) is optimized, and in the subspace membership consistency step, (4.28) is optimized. That is at each iteration t, we train the netwrok with the following algorithm.

$$\begin{cases} \text{Step 1:} \quad \min_\Theta(\mathcal{L}_{\text{S.E.}}(\mathbf{C}^t, \mathbf{Z}^t) + \lambda_1 \mathcal{L}_{\text{S.P.}}(\mathbf{C}^t) + \lambda_2 \mathcal{L}_{\text{Rec.}}(\mathbf{X}^t, \hat{\mathbf{X}}^{\mathbf{t}})), \\ \text{Step 2:} \quad \min_\Theta(\mathcal{L}_{\text{Cons.}}(\mathbf{Q}^t_{\text{Ens.}}, \mathbf{Q}^t)), \end{cases} \tag{4.29}$$

where Θ is the union of trainable parameters in the networks.

4.3.4 Finding Efficient Augmentations

In the previous section, we denoted $\mathbf{X}^t$ as a stochastic transition of $\mathbf{X}$ which is the result of applying an augmentation policy. The choice of augmentation policy plays an important role in the performance of the network. We formulate the problem of finding the best augmentation policy as a discrete search problem.

Our method consists of three components: A *score*, a *search algorithm* and a *search space* with n_s possible configurations. The search algorithm samples a data augmentation policy S_i, which has information about what image processing operation to use, the probability of using the operation in each iteration, and the magnitude of the operation. The policy S_i will be used to train a child deep subspace clustering network with a fixed architecture. The trained child network will return a score that specifies the effect of applying the policy S_i to the input data on the performance of deep subspace clustering task. Finally, all the tested policies $\{S_i\}_1^{n_s}$ will be sorted based on the returned scores.

In the following, we describe the *score* , the *search algorithm* and the *search space* in detail.

Score. In our framework, the score is a metric that evaluates the performance of the DSC on a certain given input. Note that the ground-truth labels are unknown at this stage. Therefore, we need to use a validation technique that does not use the ground-truth labels. Any internal validation of clustering methods, including mean Silhouette coefficient [196] or the Davies-Bouldin index (DBI) [231] can serve as the score metric in our search. We use mean Silhouette coefficient in this section.

Search Space. In our search space, a sample policy S_i consists of ℓ sequential sub-policies with each sub-policy using an image operation. Additionally, each operation is also associated with two hyper-parameters: 1) the probability of applying the operation, and 2) the magnitude of the operation. We discretize the range of probability and magnitude values into n_p and n_m discrete values, respectively (with uniform spacing). This way, we can use a discrete search algorithm to find them. For n_o operations, this constructs a search space with the size of $n_s = (n_o \times n_p \times n_m)^\ell$.

Search Algorithm. The size of search space n_s, can grow exponentially. A brute-force search might be impractical. To make the searching process feasible, we use a greedy search. First, we begin searching in the reduced search space where each sample policy has only one sub-policy ($\ell = 1$). In the reduced search space, we find the best probability and magnitude for each image operation. Note that n_p and n_m can also be decreased as much as necessary to keep the search tractable.

Once we find the best augmentation operations for the first sub-policy, we search for the second sub-policy ($\ell = 2$). For each found sub-policy in the first stage, we search for the best

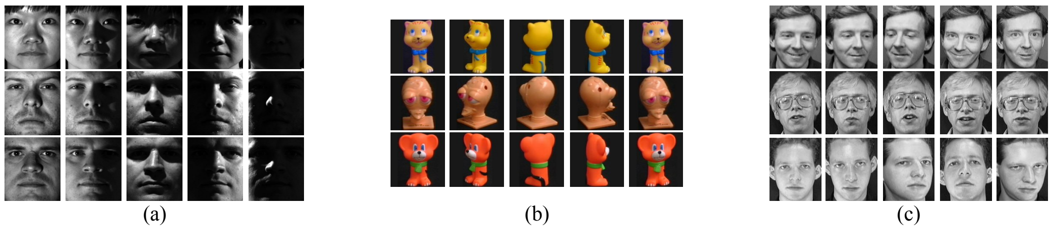


Figure 4.6: Sample images from different used datasets. (a) Extended Yale-B dataset [6]. (b) COIL dataset [7, 8] . (c) ORL dataset [9].

combination of image operations and their probabilities and magnitudes.

This process continues until we reach $\ell = \ell_{\text{max}}$, the maximum number of sub-policies. At this point, we sort all the potentially good policies that are found until this point, and select the best b augmentation policies among them.

4.3.5 Experimental Results

We evaluate our method against state-of-the-art subspace clustering algorithms on three standard datasets. We first use the algorithm described in section 4.3.4 to find the best augmentation policies for each dataset. Then, we use the found augmentation policies in the ablation study as well as in comparisons with state-of-the-art subspace clustering algorithms.

We use the following datasets in our experiments:

Extended Yale-B dataset [6] contains 2432 facial images of 38 individuals from 9 poses and under 64 illuminations settings.

ORL dataset [9] includes 400 facial images from 40 individuals. This corresponds to only 10 samples per subject.

COIL-100 [7] and **COIL-20** [8] datasets are respectively consisted from images of 100 and 20 objects placed on a motorized turnable. Per each object, 72 images are taken at pose intervals of 5 degrees that covers a 360 degrees range. Following most of the prior studies, in our experiments, we use grayscale images of these datasets.

Figure 4.6 (a), (b), and (c) show sample images from the Extended Yale-B, ORL and COIL datasets, respectively. Note that in the subspace clustering tasks, the datasets are not split into training and testing sets. Instead, all the existing samples are used in both the learning stage and the performance evaluation stage.

Experimental Setups. While our method can be applied to many DSC algorithms, unless otherwise stated, due to its promising performance, we adopt the MLRDSC networks [221] and apply our method to its networks. We call the result MLRDSC with Data Augmentation (MLRDSC-DA). The objective function of MLRDSC can be also written in the format of (4.24). The *self-expressiveness* and *subspace-preserving* loss terms in MLRDSC are

$$\mathcal{L}_{\text{S.E.}}(\mathbf{C},\mathbf{Z}) = \sum_{l=1}^{L} \|\mathbf{Z}_l - \mathbf{Z}_l(\mathbf{G}+\mathbf{D}_l)\|_F^2 \quad \text{and} \quad \mathcal{L}_{\text{S.P.}}(\mathbf{C}) = \|\mathbf{Q}^T|\mathbf{G}|\|_1 + \lambda_3 \sum_{l=1}^{L} \|\mathbf{D}_l\|_F^2, \tag{4.30}$$

where L is the number of layers in the autoencoder, $\mathbf{Z}^l$ is the features at the l-th layer, and $\mathbf{C} = \mathbf{G} + \frac{1}{L}\sum_{l=1}^{L}\mathbf{D}_l$. The coefficient matrix in this model is calculated by the consistency matrix $\mathbf{G}$ and distinctive matrices $\{\mathbf{D}_l\}_{l=1}^{L}$. The distinctive matrices enforce subspace-preserving across different layers, and $\mathbf{G}$ captures the shared information between the layers.

In the training of MLRDSC-DA, we first pre-train the networks by performing the MLRDSC algorithm. Then, we continue training MLRDSC-DA for a few additional iterations until convergence with (4.29) as

$$\begin{cases} \text{Step 1:} \quad \min_\Theta \quad \sum_{l=1}^{L} \|\mathbf{Z}_l^t - \mathbf{Z}_l^t(\mathbf{G}^t + \mathbf{D}_l^t)\|_F^2 + \lambda_1 \|\mathbf{Q}^{t^T}|\mathbf{G}^t|\|_1 \\ \qquad\qquad\qquad\qquad +\lambda_3 \sum_{l=1}^{L} \|\mathbf{D}_l^t\|_F^2 + \lambda_2 \|\mathbf{X}^t - \hat{\mathbf{X}}^{\mathbf{t}}\|_F^2, \\ \text{Step 2:} \quad \min_\Theta \quad \text{CE}(\mathbf{Q}_{\text{Ens.}}^t, \mathbf{Q}^t), \end{cases} \tag{4.31}$$

where we shape the temporal coefficient matrix as $\mathbf{C}^t = \mathbf{G}^t + \frac{1}{L}\sum_{l=1}^{L}\mathbf{D}_l^t$, and $\mathbf{Q}_{\text{Ens.}}^t$ and $\mathbf{Q}^t$ are calculated from (4.27) and (4.25), respectively.

We use the same training settings as described in [221]. This includes the same architecture for networks and values for the hyper-parameters λ_1, λ_2, λ_3 in different experiments as well as the initial values of a zero matrix for the membership matrix $\hat{\mathbf{Q}}$, and matrices with all the elements equal to 0.0001 for the coefficient matrices $\mathbf{G}^0$ and $\mathbf{D}_l^0$s at the iteration $t = 0$. We update $\hat{\mathbf{Q}}$ every 50 iterations by substituting the subspace membership estimations with the result of subspace clustering performed on the current $\mathbf{C}^t$. We set the EMA decay to $\alpha = 0.999$ in all the experiments (selected by cross-validation). We implemented our method with PyTorch. We use the adaptive momentum-based gradient descent method (ADAM) [232] with a learning rate of 10^{-3} to minimize the loss functions.

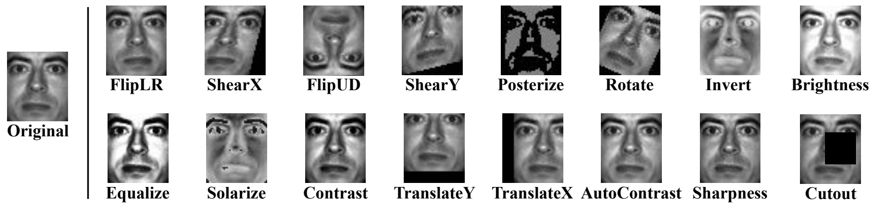


Figure 4.7: Different image transformations on a sample from the Extended Yale B dataset.

4.3.6 Best Augmentation Policies Found on the Datasets

We perform the search algorithm in Section 4.3.4 on different datasets to find the best augmentation policies for each dataset. To reduce the computations, we search in the search space of augmentation policies with the maximum number of sub-policies $\ell_{\text{max}} = 2$ (i.e. up to two sub-policies can be combined to construct a policy), and set the probability to $p = 0.1$ and the magnitude to $m = 0.3 \times r$ where $r = (\text{max}-\text{min})$ is the magnitude range that image operations accept. The image operation search space is the following set: {FlipLR, ShearX, FlipUD, SearY, Posterize, Rotate, Invert, Brightness, Equalize, Solarize, Contrast, TranslateY, TranslateX, AutoContrust, Sharpness, Cutout} that is also used in [113]. This results in a search space of $n_s = 16^2$. We selected the values for magnitude and probability of augmentation polices by searching in the full search space of augmentation policies for the first two subjects in the Extended Yale B dataset.

Figure 4.7 shows the different augmentation policies applied to a sample drawn from the Extended Yale B dataset. The details of these image operations are described in Table 1 in the supplementary materials.

For each candidate augmentation policy, we train our MLRDSC-DA model, perform subspace clustering, and return the mean Silhouette coefficient [196] as the clustering performance. We use the mean Silhouette coefficients to sort the augmentation policies and select the top two performing augmentation policies in each dataset. That is $b = 2$.

Table 4.6 shows the found augmentation policies that yield to the highest Silhouette coefficients in the subspace clustering results on different datasets. In our experiments, COIL-20 and COIL-100 resulted in similar policies. Unless otherwise stated, in all the experiments, we apply these augmentation policies to the inputs of our MLRDSC-DA algorithm.

Table 4.6: Augmentation policies that yield the highest mean Silhouette coefficient in the subspace clustering results on different datasets.

Dataset	Augmentation Policy 1	Augmentation Policy 2
Extended Yale B	(Op =‘`ShearY`’, m=0.3r, p=0.1)	(Op =‘`TranslateY`’, m=0.3r, p=0.1) + (Op =‘`Contrast`’, m=0.3r, p=0.1)
COIL-20 & COIL-100	(Op =‘`Posterize`’, m=0.3r, p=0.1) + (Op =‘`Sharpness`’, m=0.3r, p=0.1)	(Op =‘`FlipLR`’, p=0.1) + (Op =‘`Contrast`’, m=0.3r, p=0.1)
ORL	(Op =‘`ShearX`’, m=0.3r, p=0.1) + (Op =‘`Sharpness`’, m=0.3r, p=0.1)	(Op =‘`FlipLR`’, p=0.1) + (Op =‘`ShearX`’, m=0.3r, p=0.1)

Table 4.7: Ablation study of our method in terms of clustering error (%) on Extended Yale B. Top performers are bolded.

Backbone	Augmentations × Consistency Loss ×	Augmentations ✓ Consistency Loss ×	Augmentations × Consistency Loss ✓	Augmentations ✓ Consistency Loss ✓
DSC	2.67	3.10	2.56	**1.92**
MLRDSC	1.36	2.84	0.95	**0.82**

4.3.7 Ablation Study and Analysis of The Model

To understand the effects of some of our model choices, we explore some ablations of our model on the Extended Yale B dataset. In particular, we test our model on two different deep subspace clustering methods, DSC [219] and MLRDSC [221], and in four settings where 1) the consistency loss exists or 2) is ablated; 3) the optimal augmentations policies are applied to the inputs or 4) the data is fed without any augmentations.

If we remove both augmentations and the consistency loss, our networks, based on their backbones, turn to either DSC or MLRDSC networks. In the versions that data augmentation is applicable, the augmentations in Table 4.6 are used. Further analysis on the evaluation of the found augmentation policies is provided in section 4.3.9.

We report the performances in Table 4.7. As can be seen, the top performer is our full model with augmentations and the consistency loss applied to the MLRDSC method. MLRDSC-based methods, in general, outperform DSC-based methods. Consistency loss slightly improves the performance even without data augmentation. This is the result of temporal ensembling.

As can be seen in the second column of this table, applying the found augmentations to the input of DSC and MLRDSC networks without further modification (i.e., not adding the consistency loss) not only does not improve the results, but it slightly degrades the performance. These results clearly show both the importance of the consistency loss and the benefit of using data augmentations when it is combined with the consistency loss.

Table 4.8: Clustering error (%) of different methods on Extended Yale B, ORL, COIL20, and COIL100 datasets. Top performers are bolded.

dataset	LRR [215]	LRSC [233]	SSC [234]	AE+SSC [219]	KSSC [235]	SSC-OMP [236]	EDSC [237]	AE+EDSC [237]	DSC [219]	DASC [220]	S^2ConvSCN [218]	MLRDSC [221]	MLRDSC-DA Ours
E.Yale B	34.87	29.89	27.51	25.33	27.75	24.71	11.64	12.66	2.67	1.44	1.52	1.36	**0.82**
ORL	33.50	32.50	29.50	26.75	34.25	37.05	27.25	26.25	14.00	11.75	10.50	11.25	**10.25**
COIL20	30.21	31.25	14.83	22.08	24.65	29.86	14.86	14.79	5.42	3.61	2.14	2.08	**1.79**
COIL100	53.18	50.67	44.90	43.93	47.18	67.29	38.13	38.88	30.96	–	26.67	23.28	**20.67**

4.3.8 Comparison with State-of-The-Art Subspace Clustering Methods

In this section, we evaluate our method against the state of the art subspace clustering methods. We apply the found augmentation policies in Table 4.6 to the data on Extended Yale B, ORL, COIL-20 and COIL-100 datasets and feed them to our MLRDSC-DA method.

The rows in Table 4.9 report the clustering error rates of different subspace clustering algorithms. As the table reveals, deep subspace clustering methods, including DSC, ADSC, S^2ConvSCN, and ML-RDSC, in general, outperform the conventional subspace clustering approaches. This observation suggests that deep networks can better model the non-linear relationships between the samples. However, among them, our model outperforms all the benchmarks. Note that our model and MLRDSC share similar architectures and have the same number of parameters. The only difference is that our method takes advantage of training on the augmented set of data. This observation clearly shows the benefits of incorporating data augmentation in the task of deep subspace clustering.

4.3.9 Comparison with Common Augmentation Policies and Transferred Augmentation Policies

Existing automated learning algorithms for finding proper augmentations or even manual searches do not apply to the subspace clustering task. The current algorithms are mostly designed for supervised tasks and require the ground-truth targets to compare the performances, whereas, in the subspace clustering task, the ground-truth labels are not available. However, one may apply the supervised augmentation searches to a source dataset with available labels and use the found augmentation policies on a target dataset for the task of subspace clustering.

To compare such an approach with the described method in Section 4.3.4, we adopt the augmentation policies that AutoAug [113] finds on the classification task for SVHN [129] and

Table 4.9: Clustering error (%) on Extended Yale B with different augmentation policies applied to the inputs of MLRDSC-DA.

Augmentation Policies:	Random LR Flips	Cut-out	Common aug. policies	AutoAug for ImageNet	AutoAug for SVHN	Policies found from Algorithm 1 (ours)
Extended YaleB	1.32	2.88	2.96	5.96	11.31	**0.82**

ImageNet [127] datasets, and directly apply the found policies to the input of our MLRDSC-DA.

We furthermore compare the performances to the results of applying the following augmentation policies to the input: random left-right flips (*Flip-LR*), *Cut-out* [110, 238] and common augmentations picked by practitioners (*Common aug. policies*). For "Common aug. policies", we use the combination of most common augmentations, including zero paddings, cropping, random-flips, and cutout.

Note that all the experiments in this section share the same architecture and training procedure as MLRDSC-DA. They are only different in the augmentation policies that are applied to their input.

As can be seen in Table 4.9, the augmentation policies that are found with [113] on SVHN and ImageNet, perform poorly. This is because they are deemed good policies for the classification task on those datasets and may not work as efficiently on the subspace clustering task. The reason that Random Flips provides a relatively good performance is that the objects in the dataset are symmetric. The augmentations that are found with our suggested approach provide the best results.

Chapter 5

Conclusion and Future Work

In this thesis we focused on both automation and neural networks. As first, we created a new form of ANN by observing the patterns of Grid Cells. In summary, we proposed a navigational model based on biological grid cell firing patterns. The method provides a fast and memory efficient mechanism for uninformed search settings. The proposed method is useful for path-finding in unknown environments with unknown goal positions. After exploring the current methods for robot navigation, four algorithms are selected for comparisons. In the experiments, efficient implementations of comparable methods are considered. The results show higher speed (up to six times better) for GNM compared to the closest rivals among current methods. For the memory, the method usually took the second place while IDDFS used less memory than others. However, using the dynamic nature of the GNM on memory, we showed that it is possible to achieve comparatively more efficient memory usage while providing reasonable speed.

With the above method, we did show that how learning can take place with neural nets and how the computational field can connect to the field of Neuroscience. The design demonstrated the strength of automation in learning. Next we used automation to fix a major problem in data augmentation to ensure that the generated new samples cover the search space. We proposed Greedy AutoAugment as a highly efficient method to find the best augmentation policies. We combined the searching process with a simple procedure to add augmentation policies to the training data. For the experiments on the classification accuracy, we used four real datasets and eleven networks. Our results show that the proposed method could reliably improve the accuracy of classification results. The higher accuracy could be achieved while using 360 times less computational resources.

We also suggested Greedy AutoAugment for applying process that is specifically optimized for n-shot learning to extract the information from small datasets as much as possible. In the experiments, we used five prominent n-shot learning datasets. The results show the effectiveness of the proposed method, which resulted in considerable increase of accuracies for different n-shot and n-way settings. Due to its effectiveness, we expect the future integration of our proposed method with n-shot learning methods. Our experiments on Siamese and MAML neural networks represent the potential of such integrations.

In the next chapter, the goal was to ultimately present Greedy AutoAugment for unsupervised learning. Before doing that, first we showed the importance of pre-defined patterns in clustering and their effects on final results. For this purpose, we proposed a clustering process that for its underlying assumption expects separation of points in each feature instead of separation of points in multivariate space. With the new guideline, we addressed a critical problem in low-level clustering algorithms. In this regard, we used the properties of the new guideline to propose CD as a new distance metric that preserves the meaningfulness of distance values in high dimensional data. Current clustering algorithms which use Minkowski distance metrics do not provide a similar guarantee. The experiments on synthetic and non-synthetic datasets show that the proposed solution is capable of preserving meaningfulness for multivariate clustering.

In the next step, we made Silhouette a practical solution even with high number of points. Accordingly, we proposed Condensed Silhouette as an efficient version of the Silhouette score. In our design, we replaced the elements of Silhouette algorithm with equivalent inner elements of the K-Means algorithm. This helped us to maintain the accuracy of the Silhouette and at the same time, significantly reduce the computational requirements of the method. In our experiments, we used 14 real datasets that showed the effectiveness of the proposed method. In the 14 datasets, the average of the accuracy differences between Condensed Silhouette and Silhouette was only 0.0042%. For computational resources, our benchmark is WCSS, which does not need additional computations for calculating its evaluation score. In the 14 datasets, overall, Condensed Silhouette needed 5.4s, and Silhouette needed 272.73s, more time than WCSS. This result shows that Condensed Silhouette is 50.5 times closer to the benchmark than Silhouette.

At last, we used the knowledge of Chapter 4.1 and 4.2 to define Greedy AutoAugmentation for unsupervised learning. We introduced a framework to incorporate data augmentation techniques in Deep Subspace Clustering algorithms. The underlying assumption in subspace clustering tasks is that data points with the same label lie into the same subspace. Based on this assumption, we argued that slight transformations of a data point should not alter the subspace into which the data point lies. To address this property, we proposed the subspace consistency loss to keep the data points within consistent subspaces when slight random transformations are applied to the input data. Employing the mean Silhouette coefficient metric, we furthermore, provided a simple yet effective unsupervised algorithm to find the best augmentation policies for each target dataset. Our experiments showed that applying good data augmentations improves the performance oft subspace clustering algorithms.

We already demonstrated that Greedy AutoAugmentation can be used in supervised and unsupervised learning to improve the results. Specifically this method can be used to push the boundaries for normal datasets and n-shot setting. We expect that in the neat future the proposed method to improve other forms of deep neural networks such as auto-encoders, GANs, semantic segmentation and object detection. Such improvements can have positive effects in other fields such as quantifying the immunological synapse in medicine. By taking notes from these positive results, we may be able to create other forms of automation which can lead to better learning method and smarter applications. We also demonstrated that by studying improvements in neuroscience it is still possible to come by innovative ANN architectures, and improve solutions for unprecedented problems.

References

[1] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al., Matching networks for one shot learning, in: Advances in neural information processing systems, 2016, pp. 3630–3638.

[2] B. Oreshkin, P. R. López, A. Lacoste, Tadam: Task dependent adaptive metric for improved few-shot learning, in: Advances in Neural Information Processing Systems, 2018, pp. 721–731.

[3] B. M. Lake, R. Salakhutdinov, J. B. Tenenbaum, Human-level concept learning through probabilistic program induction, Science 350 (6266) (2015) 1332–1338.

[4] C. Wah, S. Branson, P. Welinder, P. Perona, S. Belongie, The Caltech-UCSD Birds-200-2011 Dataset, Tech. Rep. CNS-TR-2011-001, California Institute of Technology (2011).

[5] M.-E. Nilsback, A. Zisserman, Automated flower classification over a large number of classes, in: 2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing, IEEE, 2008, pp. 722–729.

[6] K. C. Lee, J. Ho, D. J. Kriegman, Acquiring linear subspaces for face recognition under variable lighting, IEEE Transactions on Pattern Analysis and Machine Intelligence 27 (5) (2005) 684–698.

[7] S. A. Nene, S. K. Nayar, H. Murase, et al., Columbia object image library (coil-20) (1996).

[8] S. A. Nene, S. K. Nayar, H. Murase, Columbia object image library (coil-100) (1996).

[9] F. S. Samaria, A. C. Harter, Parameterisation of a stochastic model for human face identification, in: Proceedings of 1994 IEEE Workshop on Applications of Computer Vision, IEEE, 1994, pp. 138–142.

[10] A. Naghizadeh, S. Berenjian, D. J. Margolis, D. N. Metaxas, Gnm: Gridcell navigational model, Expert Systems with Applications 148 (2020) 113217.

[11] A. Naghizadeh, M. Abavisani, D. N. Metaxas, Greedy autoaugment, Pattern Recognition Letters 138 (2020) 624–630.

[12] A. Naghizadeh, D. N. Metaxas, D. Liu, Greedy auto-augmentation for n-shot learning using deep neural networks, Neural Networks (2020).

[13] A. Naghizadeh, D. N. Metaxas, Meaningful distance for multivariate clustering, in: 2018 International Conference on Computational Science and Computational Intelligence (CSCI), IEEE, 2018, pp. 1149–1154.

[14] A. Naghizadeh, D. N. Metaxas, Condensed silhouette: An optimized filtering process for cluster selection in k-means, Procedia Computer Science 176 (2020) 205–214.

[15] M. Abavisani, V. M. Patel, Deep multimodal sparse representation-based classification, in: 2020 IEEE International Conference on Image Processing (ICIP), IEEE, 2020, pp. 773–777.

[16] M. Abavisani, A. Naghizadeh, D. N. Metaxas, V. M. Patel, Supplementary materials: Deep subspace clustering with data augmentation.

[17] T. Hafting, M. Fyhn, S. Molden, M. B. Moser, E. I. Moser, Microstructure of a spatial map in the entorhinal cortex, Nature 436 (2005) 801–806.

[18] M. Gil, M. Ancau, M. I. Schlesiger, A. Neitz, K. Allen, R. J. De Marco, H. Monyer, Impaired path integration in mice with disrupted grid cell firing, Nature Neuroscience (2017). `doi:10.1038/s41593-017-0039-3`.
URL `https://doi.org/10.1038/s41593-017-0039-3`

[19] T. Wernle, T. Waaga, M. Mørreaunet, A. Treves, M.-B. Moser, E. I. Moser, Integration of grid maps in merged environments, Nature Neuroscience (2017). `doi:10.1038/s41593-017-0036-6`.
URL `https://doi.org/10.1038/s41593-017-0036-6`

[20] M. Fyhn, T. Hafting, M. P. Witter, E. I. Moser, M. B. Moser, Gridcells in mice, Hippocampus 18 (2008) 1230–1238.

[21] J. Jacobs, C. T. Weidemann, J. F. Miller, A. Solway, J. F. Burke, X. X. Wei, N. Suthana, M. R. Sperling, A. D. Sharan, I. Fried, M. J. Kahana, Direct recordings of grid-like neuronal activity in human spatial navigation, Nature neuroscience 16 (2013) 1188–1190.

[22] M. M. Yartsev, M. P. Witter, N. Ulanovsky, Grid cells without theta oscillations in the entorhinal cortex of bats, Nature 479 (2011) 103–107.

[23] M. Fyhn, S. Molden, M. P. Witter, E. I. Moser, M. B. Moser, Spatial representation in the entorhinal cortex, Science 305 (2004) 1258–1264.

[24] N. Burgess, The 2014 nobel prize in physiology or medicine: a spatial model for cognitive neuroscience, Neuron 84 (6) (2014) 1120–1125.

[25] H. Stensola, T. Stensola, T. Solstad, K. Frøland, M. B. Moser, E. I. Moser, The entorhinal grid map is discretized, Nature 492 (2012) 72–78.

[26] C. Barry, R. Hayman, N. Burgess, K. J. Jeffery, Experience dependent rescaling of entorhinal grids, Nature neuroscience 10 (2007) 682–684.

[27] K. M. Gothard, W. E. Skaggs, B. L. McNaughton, Dynamics of mismatch correction in the hippocampal ensemble code for space: interaction between path integration and environmental cues, The Journal of neuroscience 16 (24) (1996) 8027–8040.

[28] G. Chen, J. A. King, N. Burgess, J. O'Keefe, How vision and movement combine in the hippocampal place code, Proceedings of the National Academy of Sciences 110 (1) (2013) 378–383.

[29] T. N. Schroeder, B. W. Towse, N. Burgess, C. Barry, C. F. Doeller, Optimal decision making using grid cells under spatial uncertainty, bioRxiv (2017) 166306.

[30] M. Fyhn, T. Hafting, A. Treves, M. B. Moser, E. I. Moser, Hippocampal remapping and grid realignment in entorhinal cortex, Nature 446 (2007) 190–194.

[31] E. I. Moser, E. Kropff, M. B. Moser, Place cells, grid cells and the brain spatial representation system, Neuroscience 31 (2008) 69.

[32] S. Sreenivasan, I. Fiete, Grid cells generate an analog error-correcting code for singularly precise neural computation, Nature 14 (2011) 1330–1337.

[33] A. Mathis, A. V. Herz, M. Stemmler, Optimal population codes for space: grid cells outperform place cells, Neural Computation, MIT Press 24 (9) (2012) 2280–2317.

[34] K. Gupta, U. M. Erdem, M. E. Hasselmo, Modeling of grid cell activity demonstrates in vivo entorhinal look-ahead properties, Neuroscience 247 (2013) 395–411.

[35] T. L. Bjerknes, E. I. Moser, M. B. Moser, Representation of geometric borders in the developing rat, Neuron 82 (2014) 71–78.

[36] D. Bush, C. Barry, D. Manson, N. Burgess, Using grid cells for navigation, Neuron 87 (3) (2015) 507–520.

[37] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. J. Leonard, Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age, IEEE Transactions on Robotics 32 (6) (2016) 1309–1332.

[38] D. A. Pomerleau, Alvinn: An autonomous land vehicle in a neural network, in: Advances in neural information processing systems, 1989, pp. 305–313.

[39] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE transactions on Systems Science and Cybernetics 4 (2) (1968) 100–107.

[40] X. Sun, S. Koenig, W. Yeoh, Generalized adaptive a, in: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1, International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 469–476.

[41] A. Stentz, Optimal and efficient path planning for partially-known environments, in: Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on, IEEE, 1994, pp. 3310–3317.

[42] A. Stentz, et al., The focussed d^* algorithm for real-time replanning, in: IJCAI, Vol. 95, 1995, pp. 1652–1659.

[43] S. Koenig, M. Likhachev, Fast replanning for navigation in unknown terrain, IEEE Transactions on Robotics 21 (3) (2005) 354–363.

[44] J. Antich, A. Ortiz, J. Minguez, Abug: A fast bug-derivative anytime path planner with provable suboptimality bounds, in: Advanced Robotics, 2009. ICAR 2009. International Conference on, IEEE, 2009, pp. 1–8.

[45] M. Zohaib, S. M. Pasha, N. Javaid, J. Iqbal, Iba: Intelligent bug algorithm–a novel strategy to navigate mobile robots autonomously, in: International Multi Topic Conference, Springer, 2013, pp. 291–299.

[46] H. Abelson, A. A. DiSessa, Turtle geometry: The computer as a medium for exploring mathematics, MIT press, 1986.

[47] S. Papert, Uses of Technology to Enhance Education, MIT Artificial Intelligence Memo, 1973.

[48] W. Khaksar, T. S. Hong, M. Khaksar, O. Motlagh, A fuzzy-tabu real time controller for sampling-based motion planning in unknown environment, Applied intelligence 41 (3) (2014) 870–886.

[49] U. M. Erdem, M. Hasselmo, A goal-directed spatial navigation model using forward trajectory planning based on grid cells, European Journal of Neuroscience 35 (6) (2012) 916–931.

[50] E. Rueckert, D. Kappel, D. Tanneberg, D. Pecevski, J. Peters, Recurrent spiking networks solve planning tasks, Scientific reports 6 (2016) 21142.

[51] G. Tejera, A. Barrera, M. Llofriu, A. Weitzenfeld, Solving uncertainty during robot navigation by integrating grid cell and place cell firing based on rat spatial cognition studies, in: Advanced Robotics (ICAR), 2013 16th International Conference on, IEEE, 2013, pp. 1–6.

[52] A. Banino, C. Barry, B. Uria, C. Blundell, T. Lillicrap, P. Mirowski, A. Pritzel, M. J. Chadwick, T. Degris, J. Modayil, et al., Vector-based navigation using grid-like representations in artificial agents, Nature 557 (7705) (2018) 429.

[53] E. F. Moore, The shortest path through a maze, Bell Telephone System., 1959.

[54] C. Y. Lee, An algorithm for path connections and its applications, IRE transactions on electronic computers (3) (1961) 346–365.

[55] S. W. Golomb, L. D. Baumert, Backtrack programming, Journal of the ACM 12 (4) (1965) 516–524.

[56] S. Even, Graph algorithms, Cambridge University Press, 2011.

[57] R. Sedgewick, Algorithms in C++ Graph Algorithms 3rd ed, Pearson Education, 2002.

[58] R. E. Korf, Depth first iterative deepening: An optimal admissible tree search, Artificial intelligence 27 (1) (1985) 97–109.

[59] R. E. Korf, Real-time heuristic search, Artificial intelligence 42 (2-3) (1990) 189–211.

[60] S. Koenig, Exploring unknown environments with real-time search or reinforcement learning, in: Advances in Neural Information Processing Systems, 1999, pp. 1003–1009.

[61] S. Koenig, X. Sun, Comparing real-time and incremental heuristic search for real-time situated agents, Autonomous Agents and Multi-Agent Systems 18 (3) (2009) 313–341.

[62] C. Hernández, J. A. Baier, Avoiding and escaping depressions in real-time heuristic search, Journal of Artificial Intelligence Research 43 (2012) 523–570.

[63] C. Hernández, P. Meseguer, Lrta*(k), in: Proceedings of the 19th international joint conference on Artificial intelligence, Morgan Kaufmann Publishers Inc., 2005, pp. 1238–1243.

[64] R. S. Sutton, A. G. Barto, Reinforcement learning: An introduction, Vol. 1, MIT press Cambridge, 1998.

[65] N. ALTUNTAŞ, E. Imal, N. Emanet, C. N. Öztürk, Reinforcement learning-based mobile robot navigation, Turkish Journal of Electrical Engineering & Computer Sciences 24 (3) (2016) 1747–1767.

[66] M. P. Deisenroth, G. Neumann, J. Peters, et al., A survey on policy search for robotics, Foundations and Trends in Robotics 2 (1–2) (2013) 1–142.

[67] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.

[68] M. Dorigo, T. Stützle, Ant colony optimization: overview and recent advances, in: Handbook of metaheuristics, Springer, 2010, pp. 227–263.

[69] I. Kanitscheider, I. Fiete, Training recurrent networks to generate hypotheses about how the brain solves hard navigation problems, in: Advances in Neural Information Processing Systems, 2017, pp. 4529–4538.

[70] M. J. Milford, J. Wiles, G. F. Wyeth, Solving navigational uncertainty using grid cells on robots, PLoS computational biology 6 (11) (2010) e1000995.

[71] S. Thrun, A. Bücken, Integrating grid-based and topological maps for mobile robot navigation, in: Proceedings of the National Conference on Artificial Intelligence, 1996, pp. 944–951.

[72] B. Yamauchi, A frontier-based approach for autonomous exploration, in: Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on, IEEE, 1997, pp. 146–151.

[73] J. Buford, H. Yu, E. K. Lua, P2P networking and applications, Morgan Kaufmann, 2009.

[74] A. Naghizadeh, A. Shahbahrami, Binary search routing equivalent (bsre): a circular design for structured p2p networks, Transactions on Emerging Telecommunications Technologies 28 (4) (2017) e3012.

[75] A. Naghizadeh, S. Berenjian, E. Meamari, R. E. Atani, Structural-based tunneling: preserving mutual anonymity for circular p2p networks, International Journal of Communication Systems 29 (3) (2016) 602–619.

[76] A. Naghizadeh, S. Berenjian, B. Razeghi, S. Shahanggar, N. R. Pour, Preserving receiver's anonymity for circular structured p2p networks, in: 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), IEEE, 2015, pp. 71–76.

[77] E. I. Moser, Y. Roudi, M. P. Witter, C. Kentros, T. Bonhoeffer, M.-B. Moser, Grid cells and cortical representation, Nature reviews. Neuroscience 15 (7) (2014) 466.

[78] D. Hebb, The Organization of Behavior, Wiley and Sons, 1949.

[79] E. Oja, Simplified neuron model as a principal component analyzer, Journal of mathematical biology 15 (3) (1982) 267–273.

[80] B. C. Arnold, Pareto Distribution, Wiley Online Library, 2015.

[81] P.-Y. Oudeyer, F. Kaplan, V. V. Hafner, Intrinsic motivation systems for autonomous mental development, IEEE transactions on evolutionary computation 11 (2) (2007) 265–286.

[82] K. Takaya, T. Asai, V. Kroumov, F. Smarandache, Simulation environment for mobile robots testing using ros and gazebo, in: System Theory, Control and Computing (ICSTCC), 2016 20th International Conference on, IEEE, 2016, pp. 96–101.

[83] G. Lingl, Tkinter based turtle graphics module for python, `https://docs.python.org/3.3/library/turtle.html?highlight=turtle`, accessed: 2019-08-26.

[84] M. J. Milford, G. F. Wyeth, D. Prasser, Ratslam: a hippocampal model for simultaneous localization and mapping, in: IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004, Vol. 1, IEEE, 2004, pp. 403–408.

[85] A. Gil, M. Juliá, Ó. Reinoso, Occupancy grid based graph-slam using the distance transform, surf features and sgd, Engineering Applications of Artificial Intelligence 40 (2015) 1–10.

[86] C. Cain, A. Leonessa, Fastslam using compressed occupancy grids, Journal of Sensors 2016 (2016).

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

[88] C.-C. Kuo, C.-M. Chang, K.-T. Liu, W.-K. Lin, H.-Y. Chiang, C.-W. Chung, M.-R. Ho, P.-R. Sun, R.-L. Yang, K.-T. Chen, Automation of the kidney function prediction and classification through ultrasound-based kidney imaging using deep learning, npj Digital Medicine 2 (1) (2019) 29.

[89] X. Zhu, D. Anguelov, D. Ramanan, Capturing long-tail distributions of object subcategories, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 915–922.

[90] Z. Tang, Y. Zhang, Z. Li, H. Lu, Face clustering in videos with proportion prior, in: Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.

[91] Y.-X. Wang, D. Ramanan, M. Hebert, Learning to model the tail, in: Advances in Neural Information Processing Systems, 2017, pp. 7029–7039.

[92] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, T. Brox, Discriminative unsupervised feature learning with convolutional neural networks, in: Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 27, Curran Associates, Inc., 2014, pp. 766–774.

[93] T. DeVries, G. W. Taylor, Dataset augmentation in feature space, arXiv preprint arXiv:1702.05538 (2017).

[94] Y. Bengio, G. Mesnil, Y. Dauphin, S. Rifai, Better mixing via deep representations, in: International conference on machine learning, 2013, pp. 552–560.

[95] S. Ozair, Y. Bengio, Deep directed generative autoencoders, arXiv preprint arXiv:1410.0630 (2014).

[96] X. Peng, Z. Tang, F. Yang, R. S. Feris, D. Metaxas, Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2226–2234.

[97] F. Luan, S. Paris, E. Shechtman, K. Bala, Deep photo style transfer, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4990–4998.

[98] P. Perera, M. Abavisani, V. M. Patel, In2i: Unsupervised multi-image-to-image translation using generative adversarial networks, in: International Conference on Pattern Recognition (ICPR),, 2018.

[99] M. Abavisani, V. M. Patel, Adversarial domain adaptive subspace clustering, in: Identity, Security, and Behavior Analysis (ISBA), 2018 IEEE 4th International Conference on, 2018, pp. 1–8.

[100] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, L. D. Jackel, Handwritten digit recognition with a back-propagation network, in: Advances in neural information processing systems, 1990, pp. 396–404.

[101] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[102] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[103] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[104] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 4700–4708.

[105] G. E. Hinton, A. Krizhevsky, S. D. Wang, Transforming auto-encoders, in: International Conference on Artificial Neural Networks, Springer, 2011, pp. 44–51.

[106] G. E. Hinton, S. Sabour, N. Frosst, Matrix capsules with em routing (2018).

[107] S. Sabour, N. Frosst, G. E. Hinton, Dynamic routing between capsules, in: Advances in neural information processing systems, 2017, pp. 3856–3866.

[108] Y. LeCun, The mnist database of handwritten digits, http://yann. lecun. com/exdb/mnist/ (1998).

[109] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database, in: CVPR09, 2009.

[110] T. DeVries, G. W. Taylor, Improved regularization of convolutional neural networks with cutout, arXiv preprint arXiv:1708.04552 (2017).

[111] D. Cireşan, U. Meier, J. Schmidhuber, Multi-column deep neural networks for image classification, arXiv preprint arXiv:1202.2745 (2012).

[112] G. Huang, Y. Sun, Z. Liu, D. Sedra, K. Q. Weinberger, Deep networks with stochastic depth, in: European conference on computer vision, Springer, 2016, pp. 646–661.

[113] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, Q. V. Le, Autoaugment: Learning augmentation policies from data, arXiv preprint arXiv:1805.09501 (2018).

[114] wiredfool, A. Clark, Hugo, A. Murray, A. Karpinsky, C. Gohlke, B. Crowell, D. Schmidt, A. Houghton, S. Johnson, S. Mani, J. Ware, D. Caro, S. Kossouho, E. W. Brown, A. Lee, M. Korobov, M. Górny, E. S. Santana, N. Pieuchot, O. Tonnhofer, M. Brown, B. Pierre, J. C. Abela, L. J. Solberg, F. Reyes, A. Buzanov, Y. Yu, eliempje, F. Tolf, Pillow: 3.1.0 (Jan. 2016). `doi:10.5281/zenodo.44297`.
URL `https://doi.org/10.5281/zenodo.44297`

[115] H. Inoue, Data augmentation by pairing samples for images classification, arXiv preprint arXiv:1801.02929 (2018).

[116] D. Ho, E. Liang, I. Stoica, P. Abbeel, X. Chen, Population based augmentation: Efficient learning of augmentation policy schedules, in: ICML, 2019.

[117] S. Lim, I. Kim, T. Kim, C. Kim, S. Kim, Fast autoaugment, arXiv preprint arXiv:1905.00397 (2019).

[118] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 8697–8710.

[119] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861 (2017).

[120] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2018, pp. 4510–4520.

[121] K. He, X. Zhang, S. Ren, J. Sun, Identity mappings in deep residual networks, in: European conference on computer vision, Springer, 2016, pp. 630–645.

[122] S. Xie, R. Girshick, P. Dollár, Z. Tu, K. He, Aggregated residual transformations for deep neural networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 1492–1500.

[123] X. Zhang, X. Zhou, M. Lin, J. Sun, Shufflenet: An extremely efficient convolutional neural network for mobile devices, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 6848–6856.

[124] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 116–131.

[125] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).

[126] K. Liu, Deep model infrastructures used for training gautoaugment, `https://github.com/kuangliu/pytorch-cifar`, accessed: 2019-07-26.

[127] Y. Le, X. Yang, Tiny imagenet visual recognition challenge, 2015.

[128] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, Tech. rep., Citeseer (2009).

[129] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NIPS workshop on deep learning and unsupervised feature learning, Vol. 2011, 2011, p. 5.

[130] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, Q. V. Le, Training deep models on cifar-10 and cifar-100 using autoaugment, `https://github.com/tensorflow/models/tree/master/research/autoaugment`, accessed: 2019-07-26.

[131] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv preprint arXiv:1603.04467 (2016).

[132] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, et al., Pytorch distributed: Experiences on accelerating data parallel training, arXiv preprint arXiv:2006.15704 (2020).

[133] M. Soltaniyeh, I. Kadayif, O. Ozturk, Classifying data blocks at subpage granularity with an on-chip page table to improve coherence in tiled cmps, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 37 (4) (2017) 806–819.

[134] M. Soltaniyeh, R. P. Martin, S. Nagarakatte, Synergistic cpu-fpga acceleration of sparse linear algebra, arXiv preprint arXiv:2004.13907 (2020).

[135] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.

[136] S. J. Pan, Q. Yang, A survey on transfer learning, IEEE Transactions on knowledge and data engineering 22 (10) (2009) 1345–1359.

[137] C. Szegedy, S. Ioffe, V. Vanhoucke, A. A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[138] Y. Wang, Q. Yao, Few-shot learning: A survey, arXiv preprint arXiv:1904.05046 (2019).

[139] X. J. Zhu, Semi-supervised learning literature survey, Tech. rep., University of Wisconsin-Madison Department of Computer Sciences (2005).

[140] H. He, E. A. Garcia, Learning from imbalanced data, IEEE Transactions on knowledge and data engineering 21 (9) (2009) 1263–1284.

[141] S. Hochreiter, A. S. Younger, P. R. Conwell, Learning to learn using gradient descent, in: International Conference on Artificial Neural Networks, Springer, 2001, pp. 87–94.

[142] Y. Wang, W.-L. Chao, K. Q. Weinberger, L. van der Maaten, Simpleshot: Revisiting nearest-neighbor classification for few-shot learning, arXiv preprint arXiv:1911.04623 (2019).

[143] Q. Sun, Y. Liu, T.-S. Chua, B. Schiele, Meta-transfer learning for few-shot learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 403–412.

[144] J. Snell, K. Swersky, R. Zemel, Prototypical networks for few-shot learning, in: Advances in Neural Information Processing Systems, 2017, pp. 4077–4087.

[145] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, T. M. Hospedales, Learning to compare: Relation network for few-shot learning, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1199–1208.

[146] N. Mishra, M. Rohaninejad, X. Chen, P. Abbeel, A simple neural attentive meta-learner, arXiv preprint arXiv:1707.03141 (2017).

[147] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, T. Lillicrap, Meta-learning with memory-augmented neural networks, in: International conference on machine learning, 2016, pp. 1842–1850.

[148] S. Ravi, H. Larochelle, Optimization as a model for few-shot learning (2016).

[149] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 1126–1135.

[150] R. Zhang, T. Che, Z. Ghahramani, Y. Bengio, Y. Song, Metagan: An adversarial approach to few-shot learning, in: Advances in Neural Information Processing Systems, 2018, pp. 2365–2374.

[151] Y. Lee, S. Choi, Gradient-based meta-learning with learned layerwise metric and subspace, arXiv preprint arXiv:1801.05558 (2018).

[152] E. Grant, C. Finn, S. Levine, T. Darrell, T. Griffiths, Recasting gradient-based meta-learning as hierarchical bayes, arXiv preprint arXiv:1801.08930 (2018).

[153] T. Hastie, R. Tibshirani, J. Friedman, J. Franklin, The elements of statistical learning: data mining, inference and prediction, The Mathematical Intelligencer 27 (2) (2005) 83–85.

[154] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, ImageNet Large Scale Visual Recognition Challenge, International Journal of Computer Vision (IJCV) 115 (3) (2015) 211–252. `doi:10.1007/s11263-015-0816-y`.

[155] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, M. Pontil, Bilevel programming for hyperparameter optimization and meta-learning, arXiv preprint arXiv:1806.04910 (2018).

[156] T. Munkhdalai, X. Yuan, S. Mehri, A. Trischler, Rapid adaptation with conditionally shifted neurons, arXiv preprint arXiv:1712.09926 (2017).

[157] V. Torch, Torch vision, `https://github.com/kuangliu/pytorch-cifar`, accessed: 11-14-2019.

[158] S. Chintala, Imagenet training in pytorch, `https://github.com/pytorch/examples/tree/master/imagenet`, accessed: 11-14-2019.

[159] G. Koch, R. Zemel, R. Salakhutdinov, Siamese neural networks for one-shot image recognition, in: ICML deep learning workshop, Vol. 2, Lille, 2015.

[160] D. Yiqun, Siamese neural networks for one shot image recognition implement on pytorch, `https://github.com/DuanYiqun/Siamese-Neural-Networks-for-One-shot-Image-Recognition-Implement-on-pytc` (2018).

[161] L. Long, Maml-pytorch implementation, `https://github.com/dragen1860/MAML-Pytorch` (2018).

[162] Y. Bengio, J. Louradour, R. Collobert, J. Weston, Curriculum learning, in: Proceedings of the 26th annual international conference on machine learning, ACM, 2009, pp. 41–48.

[163] Q. Sun, Y. Liu, T.-S. Chua, B. Schiele, Meta-transfer learning for few-shot learning codes, `https://github.com/yaoyao-liu/meta-transfer-learning` (2019).

[164] M. Abavisani, M. Joneidi, S. Rezaeifar, S. B. Shokouhi, A robust sparse representation based face recognition system for smartphones, in: 2015 IEEE Signal Processing in Medicine and Biology Symposium (SPMB), IEEE, 2015, pp. 1–6.

[165] M. Joneidi, A. Zaeemzadeh, S. Rezaeifar, M. Abavisani, N. Rahnavard, Lfm signal detection and estimation based on sparse representation, in: 2015 49th Annual Conference on Information Sciences and Systems (CISS), IEEE, 2015, pp. 1–5.

[166] E. Elhamifar, R. Vidal, Sparse subspace clustering: Algorithm, theory, and applications, IEEE transactions on pattern analysis and machine intelligence 35 (11) (2013) 2765–2781.

[167] M. Abavisani, V. M. Patel, Deep multimodal subspace clustering networks, IEEE Journal of Selected Topics in Signal Processing 12 (6) (2018) 1–14.

[168] Y. Wang, H. Yao, S. Zhao, Auto-encoder based dimensionality reduction, Neurocomputing 184 (2016) 232–242.

[169] F. Schroff, D. Kalenichenko, J. Philbin, Facenet: A unified embedding for face recognition and clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 815–823.

[170] D. Xu, Y. Tian, A comprehensive survey of clustering algorithms, Annals of Data Science 2 (2) (2015) 165–193.

[171] C. Hennig, T. F. Liao, How to find an appropriate clustering for mixed-type variables with application to socio-economic stratification, Journal of the Royal Statistical Society: Series C (Applied Statistics) 62 (3) (2013) 309–369.

[172] A. Naghizadeh, B. Razeghi, E. Meamari, M. Hatamian, R. E. Atani, C-trust: A trust management system to improve fairness on circular p2p networks, Peer-to-Peer Networking and Applications 9 (6) (2016) 1128–1144.

[173] A. Naghizadeh, Improving fairness in peer-to-peer networks by separating the role of seeders in network infrastructures, Turkish Journal of Electrical Engineering & Computer Sciences 24 (4) (2016) 2255–2266.

[174] A. Naghizadeh, B. Razeghi, I. Radmanesh, M. Hatamian, R. E. Atani, Z. N. Norudi, Counter attack to free-riders: Filling a security hole in bittorrent protocol, in: 2015 IEEE 12th International Conference on Networking, Sensing and Control, IEEE, 2015, pp. 128–133.

[175] R. Vidal, P. Favaro, Low rank subspace clustering (lrsc), Pattern Recognition Letters 43 (2014) 47–61.

[176] A. Y. Ng, M. I. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in: Advances in neural information processing systems, 2002, pp. 849–856.

[177] J. MacQueen, et al., Some methods for classification and analysis of multivariate observations, in: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, Vol. 1, Oakland, CA, USA., 1967, pp. 281–297.

[178] C. B. Do, S. Batzoglou, What is the expectation maximization algorithm?, Nature biotechnology 26 (8) (2008) 897–899.

[179] J. Qi, Y. Yu, L. Wang, J. Liu, K*-means: An effective and efficient k-means clustering algorithm, in: Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), 2016 IEEE International Conferences on, IEEE, 2016, pp. 242–249.

[180] J. Goldberger, S. T. Roweis, Hierarchical clustering of a mixture model, in: Advances in Neural Information Processing Systems, 2005, pp. 505–512.

[181] P. Domingos, A few useful things to know about machine learning, Communications of the ACM 55 (10) (2012) 78–87.

[182] K. Beyer, J. Goldstein, R. Ramakrishnan, U. Shaft, When is “nearest neighbor” meaningful?, in: International conference on database theory, Springer, 1999, pp. 217–235.

[183] C. C. Aggarwal, A. Hinneburg, D. A. Keim, On the surprising behavior of distance metrics in high dimensional spaces, in: ICDT, Vol. 1, Springer, 2001, pp. 420–434.

[184] N. Tomasev, M. Radovanovic, D. Mladenic, M. Ivanovic, The role of hubness in clustering high-dimensional data, IEEE Transactions on Knowledge and Data Engineering 26 (3) (2014) 739–751.

[185] R. J. Durrant, A. Kabán, When is 'nearest neighbour'meaningful: A converse theorem and implications, Journal of Complexity 25 (4) (2009) 385–397.

[186] A. Zimek, E. Schubert, H.-P. Kriegel, A survey on unsupervised outlier detection in high-dimensional numerical data, Statistical Analysis and Data Mining: The ASA Data Science Journal 5 (5) (2012) 363–387.

[187] N. Tomašev, M. Radovanović, D. Mladenić, M. Ivanović, Hubness-based clustering of high-dimensional data, in: Partitional clustering algorithms, Springer, 2015, pp. 353–386.

[188] A. Singh, A. Yadav, A. Rana, K-means with three different distance metrics, International Journal of Computer Applications 67 (10) (2013).

[189] J. A. Ramey, Collection of data sets for classification (2016).

[190] M. Lichman, UCI machine learning repository (2013).
URL `http://archive.ics.uci.edu/ml`

[191] M. E. Celebi, H. A. Kingravi, P. A. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, Expert systems with applications 40 (1) (2013) 200–210.

[192] P. Fränti, S. Sieranoja, How much can k-means be improved by using better initialization and repeats?, Pattern Recognition 93 (2019) 95–112.

[193] D. Arthur, S. Vassilvitskii, k-means++: The advantages of careful seeding, in: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[194] Z. Ansari, M. Azeem, W. Ahmed, A. V. Babu, Quantitative evaluation of performance and validity indices for clustering the web navigational sessions, arXiv preprint arXiv:1507.03340 (2015).

[195] P. J. Kaur, et al., Cluster quality based performance evaluation of hierarchical clustering method, in: 2015 1st International Conference on Next Generation Computing Technologies (NGCT), IEEE, 2015, pp. 649–653.

[196] P. J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of computational and applied mathematics 20 (1987) 53–65.

[197] C. Subbalakshmi, R. Sayal, H. Saini, Cluster validity using modified fuzzy silhouette index on large dynamic data set, in: Computational Intelligence in Data Mining, Springer, 2020, pp. 1–14.

[198] A. Lengyel, Z. Botta-Dukát, Silhouette width using generalized mean—a flexible method for assessing clustering efficiency, Ecology and Evolution 9 (23) (2019) 13231–13243.

[199] O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, I. Perona, An extensive comparative study of cluster validity indices, Pattern Recognition 46 (1) (2013) 243–256.

[200] J. Lewis, M. Ackerman, V. de Sa, Human cluster evaluation and formal quality measures: A comparative study, in: Proceedings of the Annual Meeting of the Cognitive Science Society, Vol. 34, 2012.

[201] E. Rendón, I. M. Abundez, C. Gutierrez, S. D. Zagal, A. Arizmendi, E. M. Quiroz, H. E. Arzate, A comparison of internal and external cluster validation indexes, in: Proceedings of the 5th WSEAS International Conference on Computer Engineering and Applications, 2011, pp. 158–163.

[202] Y. Liu, Z. Li, H. Xiong, X. Gao, J. Wu, S. Wu, Understanding and enhancement of internal clustering validation measures, IEEE transactions on cybernetics 43 (3) (2013) 982–994.

[203] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[204] S. Ahmadian, A. Norouzi-Fard, O. Svensson, J. Ward, Better guarantees for k-means and euclidean k-median by primal-dual algorithms, SIAM Journal on Computing (0) (2019) FOCS17–97.

[205] G. Carpaneto, P. Toth, Algorithm 548: Solution of the assignment problem [h], ACM Transactions on Mathematical Software (TOMS) 6 (1) (1980) 104–111.

[206] H. V. JOZE, M. Abavisani, Video recognition using multiple modalities, uS Patent App. 16/287,113 (May 7 2020).

[207] M. Abavisani, L. Wu, C. Davis, S. Hu, J. Tetreault, A. Jaimes, Supplementary materials: Multimodal categorization of crisis events in social media, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020.

[208] M. Abavisani, L. Wu, S. Hu, J. Tetreault, A. Jaimes, Multimodal categorization of crisis events in social media, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 14679–14689.

[209] R. Basri, D. W. Jacobs, Lambertian reflectance and linear subspaces, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 25 (2) (2003) 218–233.

[210] T. Hastie, P. Y. Simard, Metrics and models for handwritten character recognition, Statistical Science 13 (1) (1998) 54–65.

[211] M. Abavisani, V. M. Patel, Multimodal sparse and low-rank subspace clustering, Information Fusion 39 (2018) 168–177.

[212] M. Abavisani, V. M. Patel, Domain adaptive subspace clustering, in: Proceedings of the British Machine Vision Conference (BMVC), British Machine Vision Association Press, 2016, pp. 126–1.

[213] R. Vidal, Subspace clustering, IEEE Signal Processing Magazine 28 (2) (2011) 52–68. `doi:10.1109/MSP.2010.939739`.

[214] E. Elhamifar, R. Vidal, Sparse subspace clustering: Algorithm, theory, and applications, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 35 (11) (2013) 2765–2781.

[215] G. Liu, Z. Lin, Y. Yu, Robust subspace segmentation by low-rank representation, in: International Conference on Machine Learning, 2010.

[216] P. Favaro, R. Vidal, A. Ravichandran, A closed form solution to robust subspace estimation and clustering, in: IEEE Conference on Computer Vision and Pattern Recognition, 2011.

[217] V. M. Patel, H. V. Nguyen, R. Vidal, Latent space sparse and low-rank subspace clustering, IEEE Journal of Selected Topics in Signal Processing 9 (4) (2015) 691–701.

[218] J. Zhang, C.-G. Li, C. You, X. Qi, H. Zhang, J. Guo, Z. Lin, Self-supervised convolutional subspace clustering network, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 5473–5482.

[219] P. Ji, T. Zhang, H. Lia, M. Salzmann, I. Reid, Deep subspace clustering networks, in: Advances in Neural Information Processing Systems, 2017.

[220] P. Zhou, Y. Hou, J. Feng, Deep adversarial subspace clustering, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 1596–1604.

[221] M. Kheirandishfard, F. Zohrizadeh, F. Kamangar, Multi-level representation learning for deep subspace clustering, arXiv preprint arXiv:2001.08533 (2020).

[222] M. Abavisani, H. R. V. Joze, V. M. Patel, Improving the performance of unimodal dynamic hand-gesture recognition with multimodal training, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 1165–1174.

[223] M. Abavisani, V. M. Patel, Deep sparse representation-based classification, IEEE Signal Processing Letters 26 (6) (2019) 948–952.

[224] P. Y. Simard, D. Steinkraus, J. C. Platt, et al., Best practices for convolutional neural networks applied to visual document analysis., in: Icdar, Vol. 3, 2003.

[225] D. C. Cireşan, U. Meier, L. M. Gambardella, J. Schmidhuber, Deep, big, simple neural nets for handwritten digit recognition, Neural computation 22 (12) (2010) 3207–3220.

[226] X. Guo, E. Zhu, X. Liu, J. Yin, Deep embedded clustering with data augmentation, in: Asian Conference on Machine Learning, 2018, pp. 550–565.

[227] X. Guo, X. Liu, E. Zhu, X. Zhu, M. Li, X. Xu, J. Yin, Adaptive self-paced deep clustering with data augmentation, IEEE Transactions on Knowledge and Data Engineering (2019).

[228] Y. Chen, C.-G. Li, C. You, Stochastic sparse subspace clustering (2020). `arXiv:2005.01449`.

[229] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, Y. Ma, Robust recovery of subspace structures by low-rank representation, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 35 (1) (2013) 171–184.

[230] A. Y. Ng, M. I. Jordan, Y. Weiss, On spectral clustering: Analysis and an algorithm, in: Neural Information Processing Systems (NIPS), Vol. 2, 2002, pp. 849–856.

[231] D. L. Davies, D. W. Bouldin, A cluster separation measure, IEEE transactions on pattern analysis and machine intelligence (2) (1979) 224–227.

[232] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[233] R. Vidal, P. Favaro, Low rank subspace clustering (LRSC), Pattern Recognition Letters (2013).

[234] E. Elhamifar, R. Vidal, Sparse subspace clustering, in: IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 2790–2797.

[235] V. M. Patel, R. Vidal, Kernel sparse subspace clustering, in: IEEE International Conference on Image Processing, 2014.

[236] C. You, D. Robinson, R. Vidal, Scalable sparse subspace clustering by orthogonal matching pursuit, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 3918–3927.

[237] P. Ji, M. Salzmann, H. Li, Efficient dense subspace clustering, in: IEEE Winter Conference on Applications of Computer Vision, IEEE, 2014, pp. 461–468.

[238] Z. Zhong, L. Zheng, G. Kang, S. Li, Y. Yang, Random erasing data augmentation, arXiv preprint arXiv:1708.04896 (2017).