University of Cincinnati

Date: 2/22/2021

I, Xin Li, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:

ABSTRACTIVE REPRESENTATION MODELING FOR IMAGE CLASSIFICATION

Student's signature: Xin Li

This work and its defense approved by:

Committee chair: Anca Ralescu, Ph.D.

Kenneth Berman

Committee member: Kenneth Berman, Ph.D.

Kelly Cohen

Committee member: Kelly Cohen, Ph.D.

Manish Kumar

Digitally signed by Manish Kumar
DN: cn=Manish Kumar, o=University of Cincinnati, ou=Department of Mechanical Enegineering and Materials Science, email=manish.kumar@uc.edu, c=US
Date: 2021.03.02 15:13:36 -05'00'

Committee member: Manish Kumar, Ph.D.

Committee member: Anoop Sathyan, PhD

38180

Abstractive Representation Modeling

for Image Classification

A Thesis submitted to the

Division of Research and Advanced Studies of the University of Cincinnati

in partial fulfillment of the requirements for the degree of Master of Science in the Department

of Electrical Engineering & Computer Science of the

College of Engineering & Applied Science

By

Xin Li

B.S. (Accounting & Information Systems), University of Maryland, College Park

2021

Master's Examination Committee Chair:

Dr. Kelly Cohen

Dr. Anca Ralescu

Dr. Anoop Sathyan

Dr. Kenneth Berman

Dr. Manish Kumar

Dedication

This research is dedicated to my beloved parents, Rongtao Li and Juan Xin, who have given me all their supports and trust at all times.

Acknowledgements

Throughout my thesis research and writing, I received enormous support and assistance. I want to express my deepest thanks to my advisor, Dr. Kelly Cohen, who advised and supported me at his best. Working with him granted me the opportunities to explore and develop my understanding of artificial intelligence, which leads to this thesis.

Also, I would like to thank Dr. Anca Ralescu, Dr. Anoop Sathyan, Dr. Kenneth Berman and Dr. Manish Kumar for accepting to be on my committee and for their time and efforts to review my work.

Abstract

In recent years, artificial intelligence has achieved remarkable progress by showing its applicable values in various industries. Convolutional neural networks (CNN) and their derivative approaches are well known for their robustness and accuracy in handling visual data. However, as a data-driven approach, CNN also has limitations. In exchange for good performance, CNN requires a large amount of training data and a heavy training process. The intricate neural network layer design also needs to be reconstructed and tuned by experienced researchers for different problems. Finally, the "curse of Blackbox" is a well-known disadvantage of the neural network, preventing CNN from providing a reasonable explanation for the prediction results. All the above limitations remind us that the most cutting-edge approach is still in the state of weak AI. This thesis proposes an approach called Abstractive Representation Model (ARM), which is different from the traditional data-driven approaches. This goal of experimenting with such a model is to address CNN's weaknesses and possibly develop a new way of handling image data.

Keywords: Image Classification, Convolutional Neural Network, Convolutional Filter, Max-pooling, K-Means Clustering, Explainability, Abstraction.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
ARM	Abstractive Representation Model
CNN	Convolutional Neural Network
IQR	Interquartile Range
NLP	Natural Language Processing
ReLu	Rectified Linear Unit

CHAPTER 1: Introduction

1.1 Background

During recent decades, a machine learning method called deep neural networks (DNN) [1], inspired by artificial neural networks (ANN) [2], had been developed. Unlike conventional machine-learning techniques that struggled to process raw data, deep-learning [1] methods employ DNN with representation-learning approaches [3] to extract representations or features needed for classification.

One of the most popular DNNs is the convolutional neural network (CNN) [4], which became the most commonly used approach for analyzing visual imagery and natural language processing (NLP). Such data is usually involved with a high dimension, leading to the curse of dimensionality, and the resulted computation complexity issue is one of the significant limitations for traditional ANNs to compute image data. Instead of directly processing the raw data, CNN encodes image-specific features into architecture, making the network more suitable for image-focused tasks and further reducing the model's parameter requirements.

1.2 Identified Research Problems

Although it is widely used for various applications with notable feedbacks, CNN is viewed as a double-edged sword in its applications.

First of all, as a derivative approach of DNN, CNN requires enormous amounts of data in the training process so that the weights and biases of neurons in hidden layers and the pattern filters can be fully tuned for high accuracy. Also, the neural network layer design [5] has always

been a problem without an exact solution, and a well-designed network usually requires problem-related experience and time of tuning. Last but not least, although it uses representation learning to avoid the direct processing of raw data by utilizing abstractive features, CNN cannot provide explanations to its results like all other ANN-related approaches, which is well known as "black box."

Again, all these weaknesses indicate that CNN is still in a state of "weak AI" [6] despite its good performance. For example, CNN can be thought of as a student who received a high exam score. However, it can not explain its answers to the questions. Instead of genuinely understanding the knowledge, it spent a long time to "learn" by memorized almost all the answers to the questions.

1.3 Motivation

The computer has a much stronger computational ability than humans; however, the human is undoubted about having much stronger intelligence than the computer. As humans, we know how to use alternative ways to atone our limitations, whereas computers can not do the same. For example, it is usually difficult for us to understand a massive numerical data table. However, by plotting the data into a graph, we can easily extract the information from it. The human brain has an impressive ability to process visual data. The actual views that human percepts through eyes contain huge information in pixel values. However, what humans perceive are not the pixel values but the abstractive understanding of the views that are converted by brain activities. CNN's representation learning technique simulates similar logic by converting raw image data into abstractive features through multiple neural network hidden layers. However, CNN does not further develop correlations among the abstractive features to explainability.

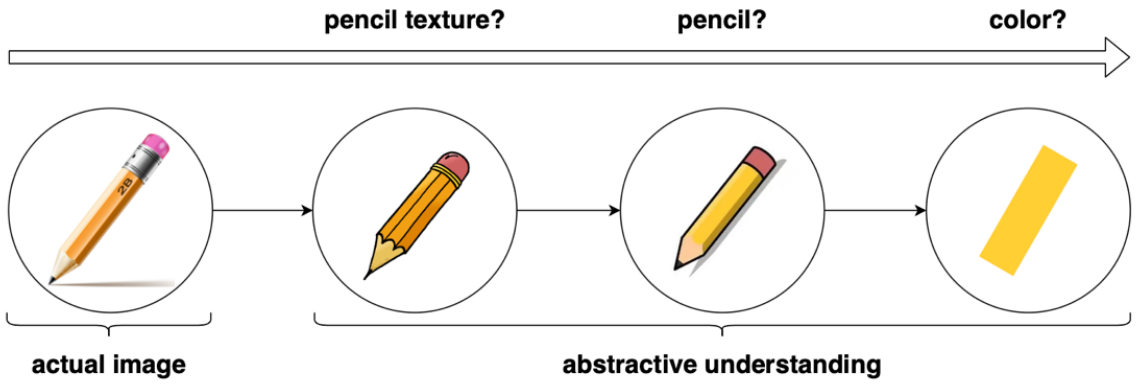

Figure 1.1 Abstractive representations of different objectives.

To better understanding how humans process visual data, we use an illustration of perceiving a pencil, as shown in Figure 1.1. Assumed the most left pencil is what the eyes captured, and the right three pencils are abstractive understandings of what humans perceived in the brain. Those three abstractive understandings of the same pencil contain different combinations of details. As soon as an object is perceived, the brain decides what and how much information to be extracted from the object's original view. Some of the information, such as the pencil's color and the pencil's shape, is used multiple times with other information to construct different understandings. As the objective of recognition changes (see Figure. 1.1), a new abstractive representation composed of more or less extracted information is generated in the head through abstractive processing. Such a process requires a systematic and hierarchical design to represent a feature as a composition of other features.

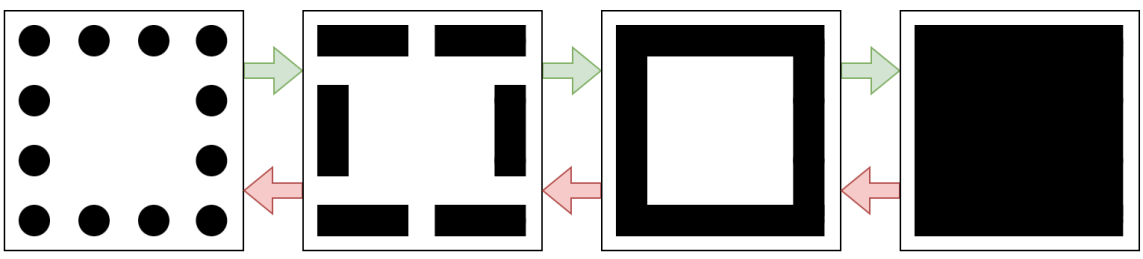

Figure 1.2 Example of bidirectional abstractive processing.

An example of abstractive processing is shown in Figure 1.2, where information is selected and grouped bidirectionally into new representations so that different requirements of understanding about the image can be fulfilled. A similar philosophy is employed in this paper, in which not only features are extracted but also correlations among features and derivative features are established. We assumed, using abstractive representations generated from a few samples can also perform the same classification tasks.

1.4 Solution Overview

In this research, the Abstractive Representation Model (ARM) is proposed, and it attempts to address the limitations of CNN by exploring a different perspective of handling image data.

Two significant parts compose ARM. The first part focuses on data processing which turns raw image data into abstractive representations for different abstraction levels. The representations in each abstraction level can be viewed as the transformed version of the original pixel-related image so that the image data is replaced and will not be used in later processes. Researchers have the freedom of either optimizing the ARM with training data or "teaching" the ARM directly by modifying ARs manually. More details about the "teaching mode" of ARM are introduced later. ARM is designed to eliminate the difficulties of "layer structure design" using no neural network. Also, since ARM focuses on understanding and explaining the images with abstractive representations instead of using an experiential learning process, it does not require extensive input data for training like other data-driven approaches.

The second part of ARM focuses on using abstractive representations instead of training directly with data in pixel values for classification. In this paper, our primary focus is on introducing how images are interpreted by ARM in abstractive representations. K-Means clustering [7] is chosen for classification in a simple implementation perspective. Of course,

other well-known classification methods such as the decision-tree and Bayesian approach should be considered for future research. Decision-tree's purity or entropy functions are critical for considering it as a right clustering approach for ARM [8]. ARM faces optimization issues at the current development stage, which can be solved either by training with a large amount of data or performing clustering over "meaningless" rules. The Bayesian approach [9] is chosen for a similar reason since it analyzes the relationship between independent and dependent variables so that the conditional probability is derived for each relationship. Such relationships are viewed as the same as the "rules" in ARM.

CHAPTER 2: Literature Review

The abstractive representation model (ARM) is inspired by multiple techniques used by convolutional neural networks (CNN), and these techniques will be reviewed in this section to understand the ARM implementation better. Also, K-Means clustering will be introduced for classification.

2.1 Convolutional Neural Network

The fundamental architecture of CNN comprises three main types of layers: convolutional layers, pooling layers, and fully-connected layers, and a simple architecture example for the classification over the MNIST dataset is shown in Figure 2.1 [10].

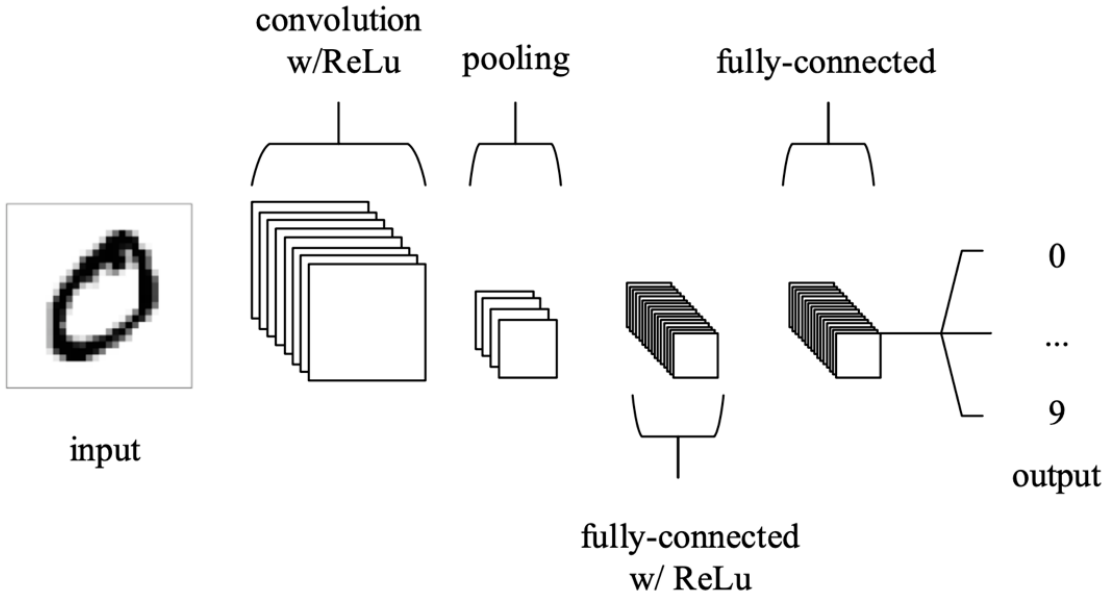

Figure 2.1 Simple CNN architecture.

The above architecture can be broken down into the input layer, convolutional layer, pooling layer, and fully-connected layer. The input layer contains the input image's pixel values, ready for further processing. The convolutional layer determines neurons' output by extracting the original image's local features through feature filters. The pooling layer is used for down-

sampling along with the given input's spatial dimensionality so that the number of parameters in the future calculation is further reduced. The fully-connected layer of CNN serves the same role in ANNs, producing class scores for classification.

ARM introduced in this research will utilize the first three layers of CNN: the input layer, convolutional layer, and pooling layer. The input layer and pooling layer in ARM and CNN are implemented similarly, which provides the original image's pixel values for the following layers. The main differences between ARM and CNN are revealed in the convolutional layer [10].

The key element within the convolutional layer is called a kernel, which can be understood as a feature filter. In convolutional layers, raw image data is convolved by each filter across the input's spatial dimensionality to produce a 2D activation map. A comparison between CNN and ARM in kernel and activation perspective is illustrated in Figure 2.2. For CNN, kernels are initialized randomly, and they are tuned during training. However, the kernels used by ARM are fixed. In both approaches, each kernel is associated with an activation map that collects the full output volume from the convolutional layer. CNN's activation function is called rectified linear unit (ReLu) [10], forcing negative values to zeros. ARM uses a binary activation that outputs 1 for identical match and 0 otherwise.

CNN utilizes pooling layers to gradually reduce the representation's dimensionality, which results in reduced computational complexity of the model. The pooling layer is used in ARM not only for the same reason but also for providing abstractive meaning. As shown in Figure 2.3, max-pooling [10] is commonly used in the pooling layer, and it uses the "MAX" function to scale the activation map inputted by the convolutional layer.

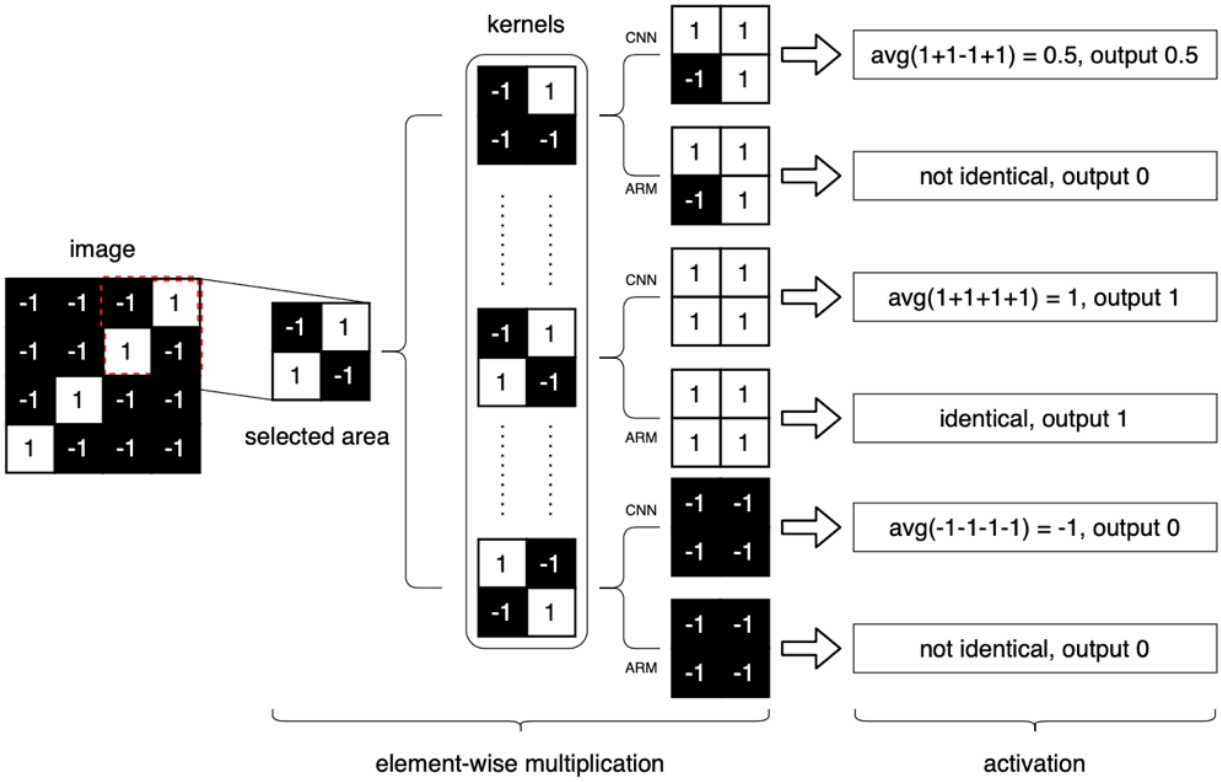

Figure 2.2 Difference of kernel applications between CNN and ARM.

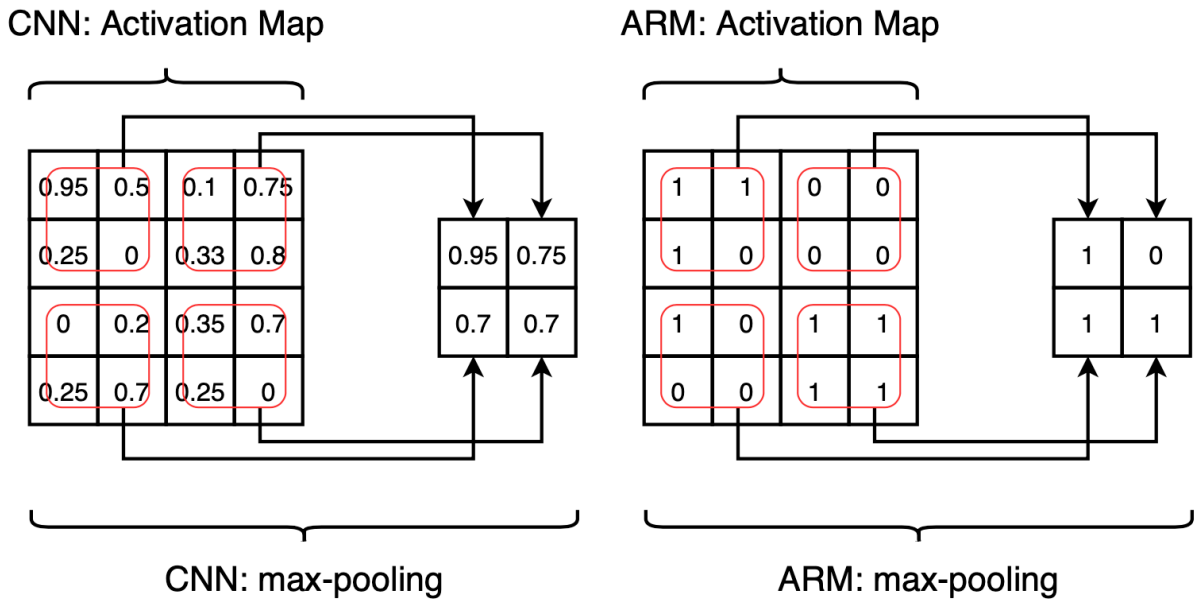

Figure 2.3 Difference of max-poolings between CNN and ARM.

2.2 K-Means Clustering

K-Means clustering is a commonly used algorithm for classifying or grouping data based on the features or attributes into *K* number of clusters. *K* is a positive integer that indicates the number of expected clusters to be found. The clustering is done by minimizing the sum of squared Euclidean distances [7] between data and the corresponding cluster centroid. The Euclidean distance between two data point is calculated as shown in Equation (2.1),

$$d(p,q) = \| p - q \| = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \tag{2.1}$$

where *d* is the number of attributes of data. The traditional K-Means clustering is usually involved with optimization to minimizing the sum of squared errors (SSE) [7] as shown in Equation (2.2),

$$SSE(C) = \sum_{i=1}^{k} \sum_{x_j \in c_i} \left(x_j - \mu_i\right)^2 \tag{2.2}$$

where μ_i and x_j are the mean and data point of cluster c_i.

However, K-Means clustering is used by ARM differently. The data points for K-Means to perform clustering within ARM are the pattern binary detector introduced in the following chapters. The clusters are also already known, and the ground truth cluster centroids since ARM is implemented as a supervised learning method. Thus, ARM does not need optimization to minimize SSE like K-Means clustering. Instead, the clustering or classification in ARM is done by only discovering the cluster with a centroid closest to the target data.

CHAPTER 3: Data

3.1 MNIST Digit Dataset

Since abstractive representation modeling (ARM) is currently in an early stage of development, for simplicity, the choices of data used for training and testing should be lightweight. As a result, we can evaluate ARM's general performance and explainability.

Figure 3.1 Examples of MNIST digits.

As a result, the MNIST digit dataset [11] is used since it is a popular choice for examining new image classification approaches. Before converting the image into abstractive representation, the MNIST data is pre-processed into a binary version to reduce image texture complexity. Such pre-processing will lose information of the images; however, texture information trivially affects the result for the digit classification problem. For future work, a more complex dataset will be needed for ARM, which has texture information been considered for real-life problems such as face recognition.

The MNIST digit dataset contains handwriting digits range from 0 to 9 (see Figure 3.1). The dataset is divided into training and testing sub-datasets, and the dataset batches are prepared by randomly selecting data from each sub-datasets with pre-defined sizes, as shown in Table 3.1.

Table 3.1 Dataset Batches.

Training Dataset Size	10	50	100	150	200	250	300	1000
Testing Dataset Size	500	500	500	500	500	500	500	500

CHAPTER 4: Method

4.1 Abstractive Processing

The raw image data is normally presented in pixel values, which are computationally expensive to process and difficult to interpret. In the abstractive representation modeling (ARM) approach, the image data is converted from pixel form through abstractive processing into a new form, called abstractive representations, composed of lists of rules.

The building blocks of abstractive representation are basic patterns and layout patterns. Basic patterns are pixel-level patterns containing the most fundamental pixel patterns without any abstractive processing. Basic pattern filters are applied only once at the beginning of the entire system processing to extract the next processing stage's basic patterns. Layout patterns reveal the layouts composed of basic patterns in the image. Layout pattern filters will be used multiple times depends on the depth of pre-defined abstractive levels.

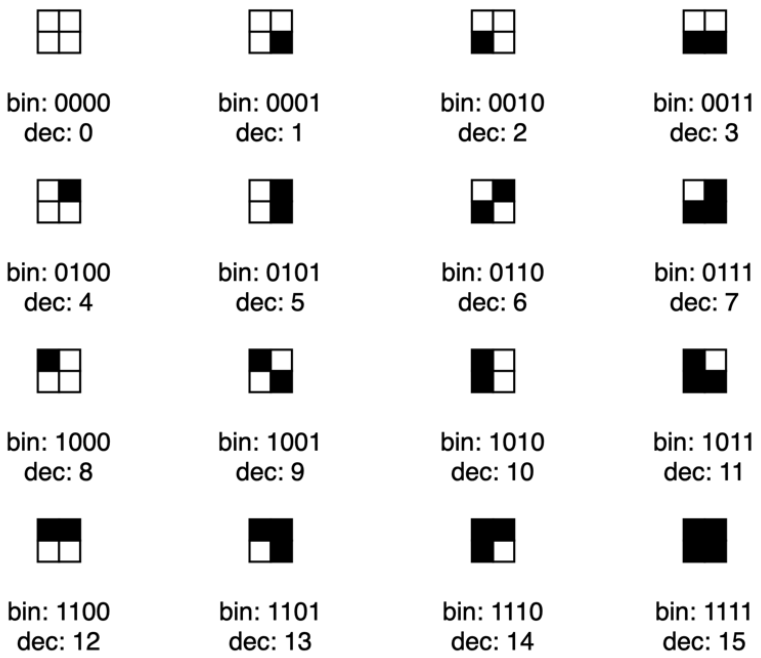

Figure 4.1 Basic and layout pattern filter definition.

Both basic and layout pattern filters need to be initialized with a size of *N* x *N*, and they can be defined in different sizes depends on the problems. This paper considers the complexity of the MNIST digit dataset and computational cost. A size of 2 x 2 is defined for both basic and layout pattern filters (see Figure 4.1). Binary numbers construct such pattern filters in 2 x 2 matrix format with corresponding decimal numbers, representing the rule matrix's indexes.

Abstractive processing is implemented with multiple abstractive levels, and each level consists of convolutional filtering and max-pooling. An illustration of abstractive processing from abstractive level-0 to level-1 is shown in Figure 4.2, where (a) and (b) are the basic pattern filtering over input image and max-pooling on basic pattern results. (c) in Figure 4.2 is the layout pattern filtering over basic pattern max-pooled results. Convolutional filtering ensures the local detail features can be extracted, and max-pooling abstracts the image features by discarding and reserving information.

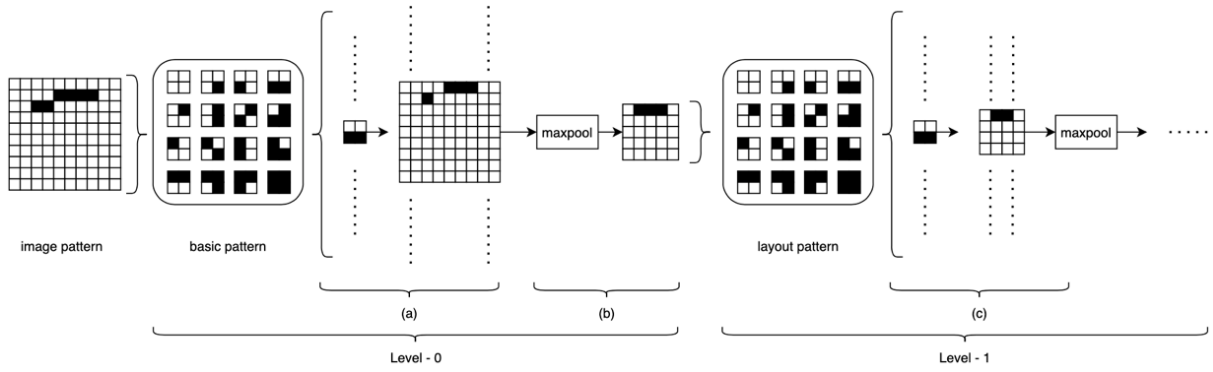

Figure 4.2 Abstractive processing.

At abstractive level-0, the original input image in a shape of *N* x *N* is filtered convolutionally by 16 basic pattern filters. This level's outputs are 16 binary matrices of size *N*-1 x *N*-1, which reveal each pattern's locations in the input image. Then, the 16 binary matrices are max-pooled individually to reduce output matrices' sizes. Max-pooling is used for merging patterns in a given scope. For example, one horizontal pattern is different from 3 horizontal

patterns for sure from the pixel perspective. However, they can be considered the same pattern within the scope of 2 x 2 from a higher abstractive point of view. The max-pooling scope can be defined with different sizes, and the larger size indicates a higher degree of abstraction, vice versa. In this paper, the max-pooling scope is defined as 2 x 2, and 16 matrices of shape (*N*-1)/2 x (*N*-1)/2 are the outputs that will be used for the next level of abstractive processing.

4.2 Abstractive Level Design

The ARM's architecture is designed with levels, and each level outputs a level-specific understanding of the original image. Shallower levels tend to have more details of the original image with less abstractive understandings. On the other hand, deeper levels will lose more details of the original image to better abstract the image. As a result, an image's overall understanding is eventually composed of the outputs from all abstractive levels.

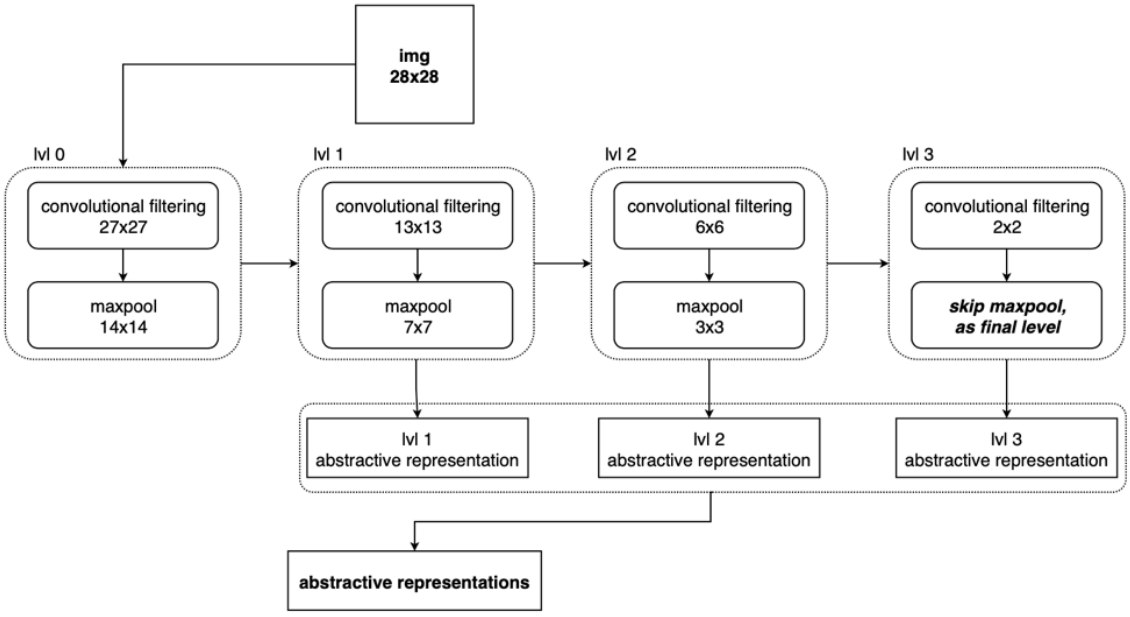

Figure 4.3 Abstractive level design.

In this paper, the abstractive levels are defined in a way shown in Figure 4.3, which has one initial level for basic pattern extraction and three abstractive levels for layout pattern

extraction. The initial level is always for basic pattern extraction so that later levels can continue from the initial level outputs. Then, the remaining abstractive levels are all about layout pattern extractions. Two criteria need to be considered for a reasonable level design. First of all, given the size of the input image, convolutional filters, and max-pooling size are defined, the depth of level needs to ensure the final level's output is at least in a size of 2 x 2 or larger. Because the abstractive representation outputted from each level becomes smaller in size, a profound level will result in a meaningless representation, such as the size of 1 x 1. The second criteria are the tradeoff between accuracy and computational cost. More abstractive meaning can be extracted from the image as the level progresses deeper but with exponentially increasing computations. Thus, if the objective is satisfied by a certain depth, a more in-depth level design is unnecessary.

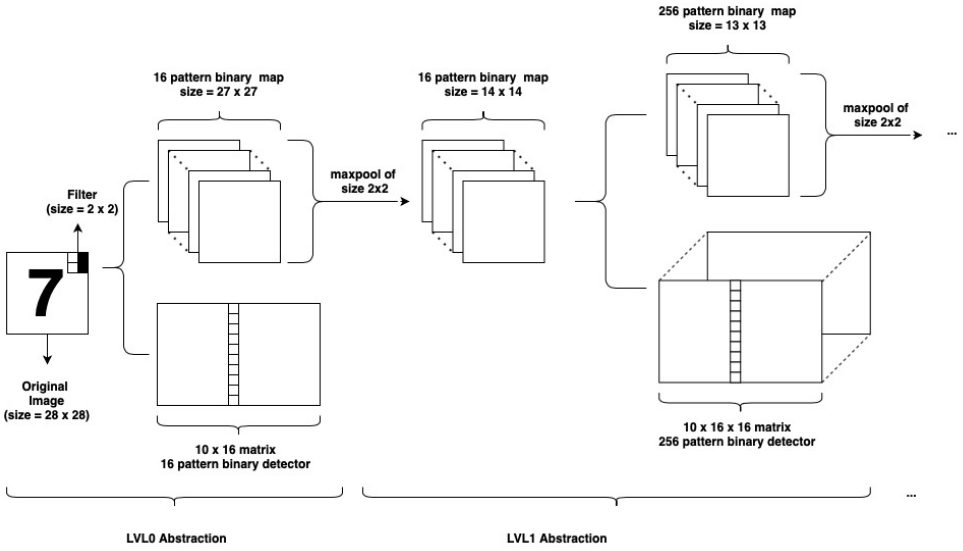

Figure 4.4 Details of abstractive processing.

More details of each abstractive level are revealed and illustrated in Figure 4.4. At each level, there are two outputs produced, which are pattern binary map and pattern binary detector. Pattern binary map indicates the detected locations of a particular pattern, and such information will be used as the input of the next level after the max-pooling. On the other hand, the pattern

binary detector is a matrix that indicates the encounters of all patterns at the current level, and each cell represents a rule that describes a unique pattern. As collections of rules, the pattern binary detectors will construct the abstractive representations for prediction.

The system-level design is naïve and straightforward at the current development stage, which is only suitable for laboratory experiments instead of real-life applications. Several limitations should be resolved in the future. First of all, the layout pattern extraction is based on the initial 16 patterns individually without any interactions among patterns. Also, the computational cost exponentially increases as the level progresses deeper. A better data structure should be designed to solve such issues.

4.3 Pattern Frequency Detector

Since there is no optimization implemented in ARM, each input image is only processed once during the training process. It is different from the traditional learning algorithms that use "epochs and batches" to iterate through the training data. After the entire training dataset is processed through all the abstractive levels, a rule summarization is applied to the pattern binary detectors to produce pattern frequency detectors for each digit.

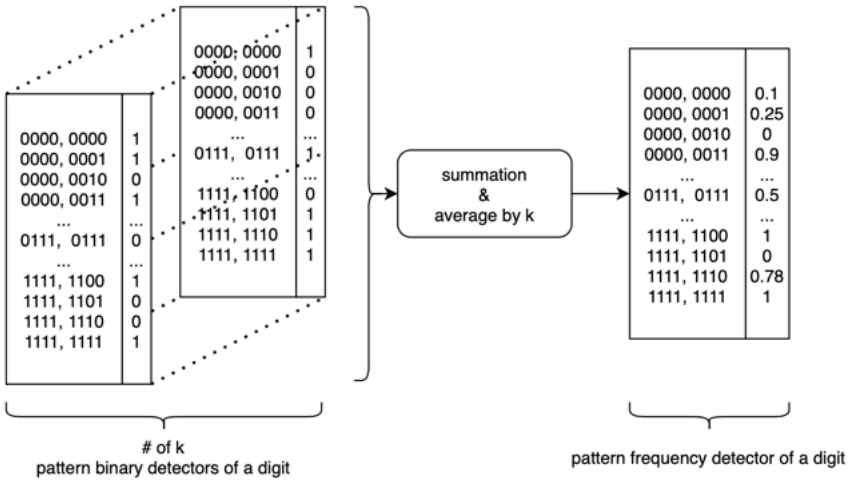

Figure 4.5 Level-1 rule summarization for a single digit.

In Figure 4.5, *k* is the number of pattern binary detectors, and it is defined by the number of training images given per class. The summation and averaging are applied to all binary detectors class-wisely to retrieve a single pattern frequency detector for each digit. Such rule summarization and averaging are processed through all levels, and the final outputs are three pattern frequency detectors in the form of matrices. The frequency detector indexes represent the rules, and the values in cells indicate the degree of confidence about the rules. Each frequency detector is considered an abstractive representation of the corresponding abstractive level, and a collection of these detectors can then be used for classification.

4.4 Classifier

The classifier is designed using K-Means clustering with weight and bias assigned to each abstractive level. Pattern frequency detector can be considered the mean of each class, and Euclidean distance is used to measure the similarity between testing image and frequency detectors. The weight and bias are used to adjust the impacts of different abstractive levels to the classification. Such weights and biases need to be pre-defined to give more credits to a more profound level.

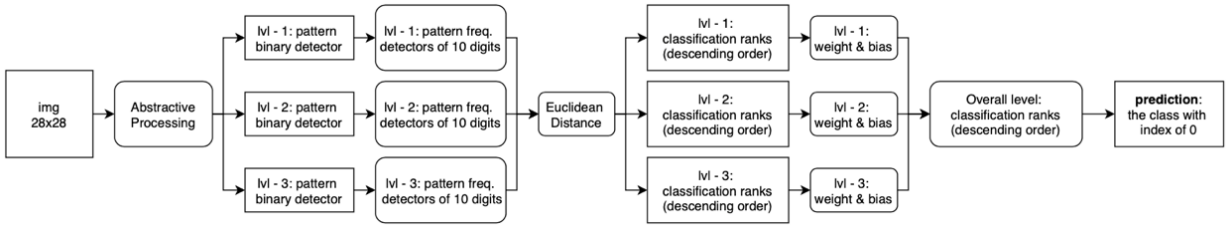

Figure 4.6 ARM's K-Means classifier.

An illustration of the K-Means classifier is shown in Figure 4.6. Three pattern binary detectors are extracted from the testing image through abstractive processing. These detectors are then paired with pattern frequency detectors level-wisely to calculate the Euclidean distances

class-wisely. The resulting 10 Euclidean distances indicate the similarities between the unknown testing image and ten classes. Such similarities are further processed with weights and biases to retrieve the class ranks for each level. Finally, the final class rank results from combining and sorting the local class ranks of all levels, and the class with the top rank is the predicted class of the testing image.

CHAPTER 5: Experimental Results

Three experiments are implemented in this chapter to examine abstractive representation modeling's (ARM) performance. Since ARM is inspired by the convolutional neural networks (CNN), one of the most popular image classification approaches, CNN is chosen to compare with ARM. First, a parallel comparison between ARM and CNN is made to test the overall performance with different training data sizes. Then ARM is tested with different sizes of training data again, but this experiment focuses on the local performance, which is the accuracy of predicting each class. Finally, a robustness test is made to compare ARM and CNN's stability after training with small and large data.

5.1 Global Performance Analysis of ARM vs. CNN

This experiment focuses on evaluating ARM's performance under different training data sizes.

Table 5.1 Global Performance Comparison of ARM and CNN.

model	learning rate	batch size	training size	iteration	total images processed	testing size	accuracy
ARM	N/A	N/A	10	1	10	500	0.302
CNN	0.001	5	10	100	1000	500	0.38
ARM	N/A	N/A	50	1	50	500	0.7
CNN	0.001	10	50	100	5000	500	0.714
ARM	N/A	N/A	100	1	100	500	0.77
CNN	0.001	10	100	100	10000	500	0.796
ARM	N/A	N/A	150	1	150	500	0.844
CNN	0.001	10	150	100	15000	500	0.854
ARM	N/A	N/A	200	1	200	500	0.854
CNN	0.001	10	200	100	20000	500	0.912
ARM	N/A	N/A	250	1	250	500	0.85
CNN	0.001	25	250	100	25000	500	0.922
ARM	N/A	N/A	300	1	300	500	0.896
CNN	0.001	15	300	100	30000	500	0.936
ARM	N/A	N/A	1000	1	1000	500	0.87
CNN	0.001	20	1000	100	100000	500	0.958

As shown in Table 5.1, both models are given 10, 50, 100, 150, 200, 250, 300, and 1000 images during the training process. All training images are chosen to have 1, 5, 10, 15, 20, 25, 30, and 100 images for each class to ensure proper training for all classes. Because ARM does not implement an optimization, it only requires one iteration, which means all training data is processed only once. In contrast, for the MNIST digit dataset, CNN needs about 100 iterations to achieve convergence. The testing images are chosen similarly as training images, in which each class has 50 images for prediction. Because the training data varies in size, the overall testing dataset sizes are fixed to be 500 to eliminate potential bias in the testing stage when comparing performance across different training.

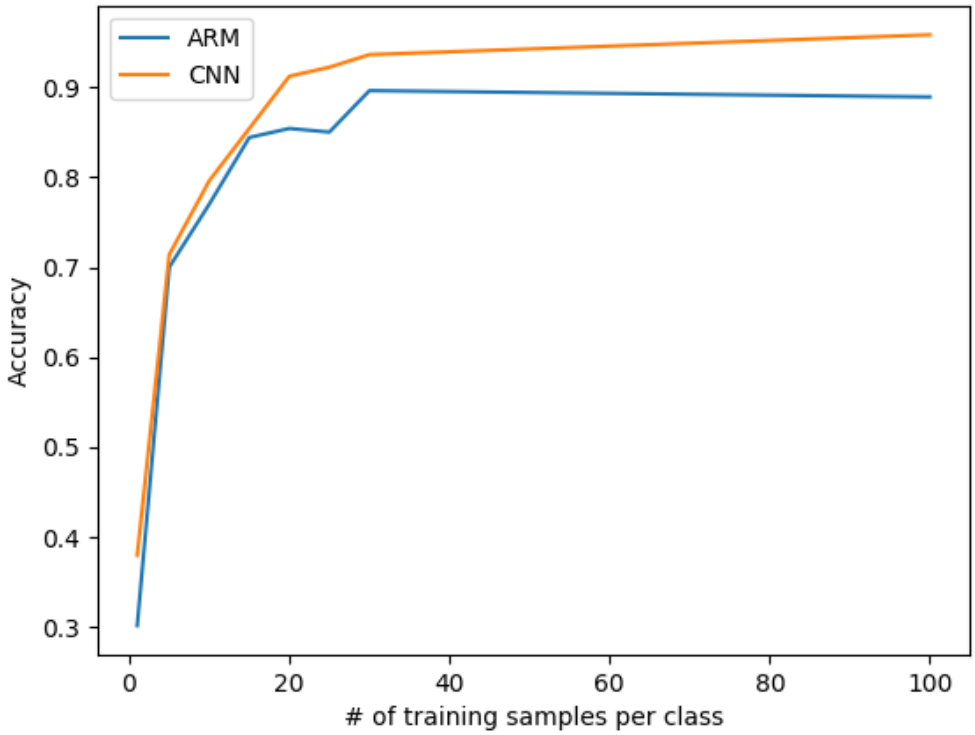

Figure 5.1 Global Performance of ARM and CNN.

In Figure 5.1, ARM received similar accuracies as CNN when it is trained given 1, 5, 10, and 15 images per class. After that, the more training data is given, the better performance CNN made. ARM's accuracies stay around 85% to 90% at best, which is expected and acceptable.

ARM is expected to be better than CNN when training with a limited amount of data and gradually loses its advantage as more and more data is provided. Because there is no optimization implemented for ARM, the classifiers have many noisy rules that intervene in the decision-making process. However, even with such a non-optimized model, ARM achieved similar accuracy as CNN with few training data and converged at the accuracies of 85% and above. It indicates that ARM can outperform CNN if optimization is implemented to eliminate the noisy rules.

5.2 Local Performance Analysis of ARM

Given different sizes of training data, this experiment focuses on evaluating ARM's performance of classifying digits with different levels of complexities in patterns. Again, as in the previous experiment, 1, 5, 10, 15, 20, 25, and 30 images for each class are given as training data, and each trained model is required to make predictions on 500 testing data. In this experiment, the overall performance is generated and the accuracy of predicting each digit individually.

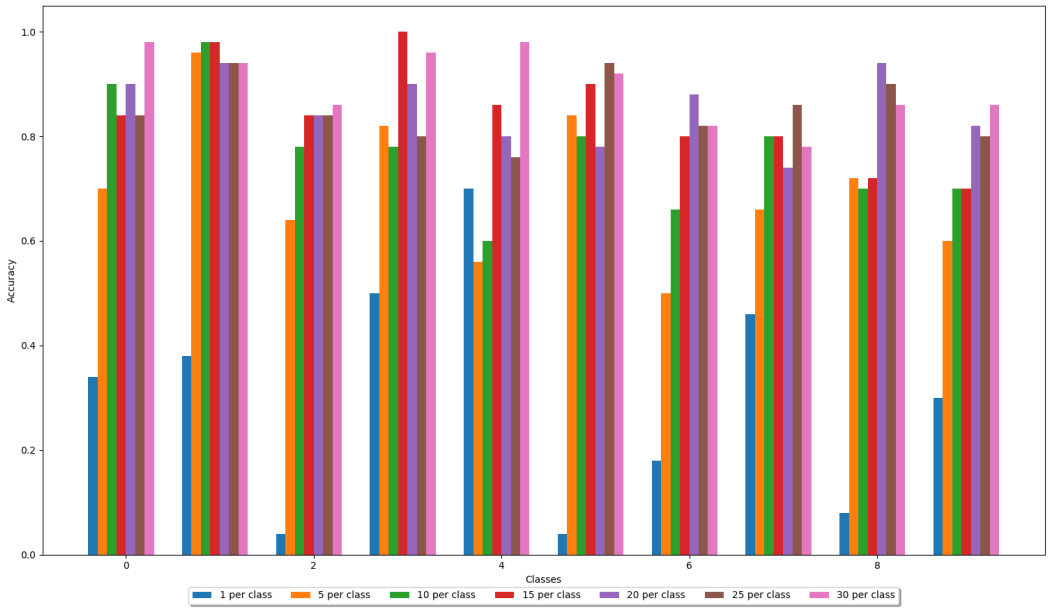

Figure 5.2 Class-wise performance of ARM.

As shown in Figure 5.2, except for the model trained by the data of 1 image per class, all other ARM models have relatively stable accuracies without colossal variance. Simple digits such as 0, 1, and 2 have much stabler accuracies than complicated digits such as 3 and 4. It means ARM requires optimizing rules to handle complicated patterns better, even though the overall performance is acceptable.

5.3 Robustness Analysis of ARM vs. CNN

Because ARM is not a data-driven approach, robustness is a critical aspect to be evaluated. Unlike CNN, which achieves outstanding robustness and high accuracy by optimizing the model through a neural network, ARM only processes the training data once. This experiment evaluates ARM's stability using a small amount of training data.

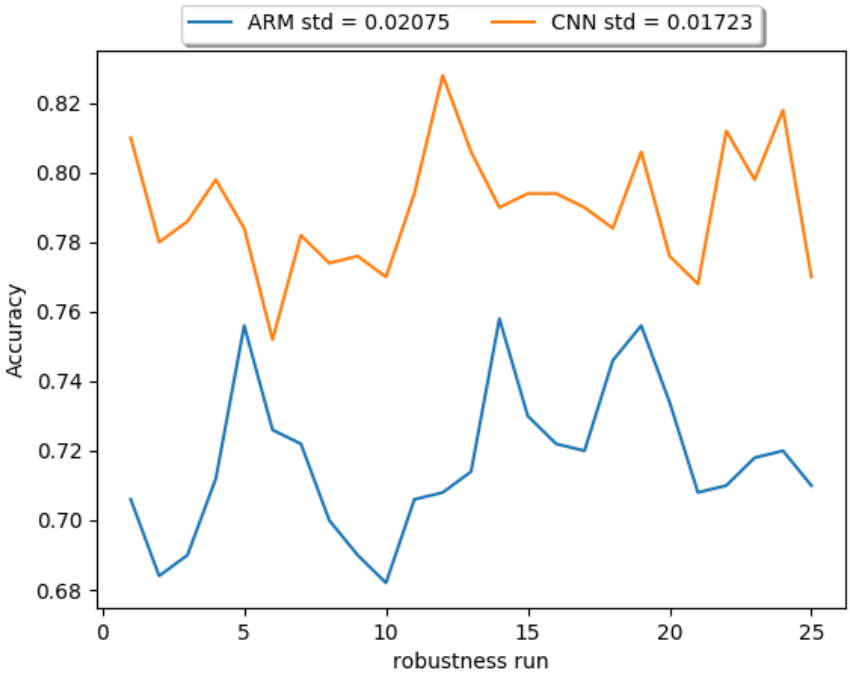

Figure 5.3 Performance comparison using 5 training images per class.

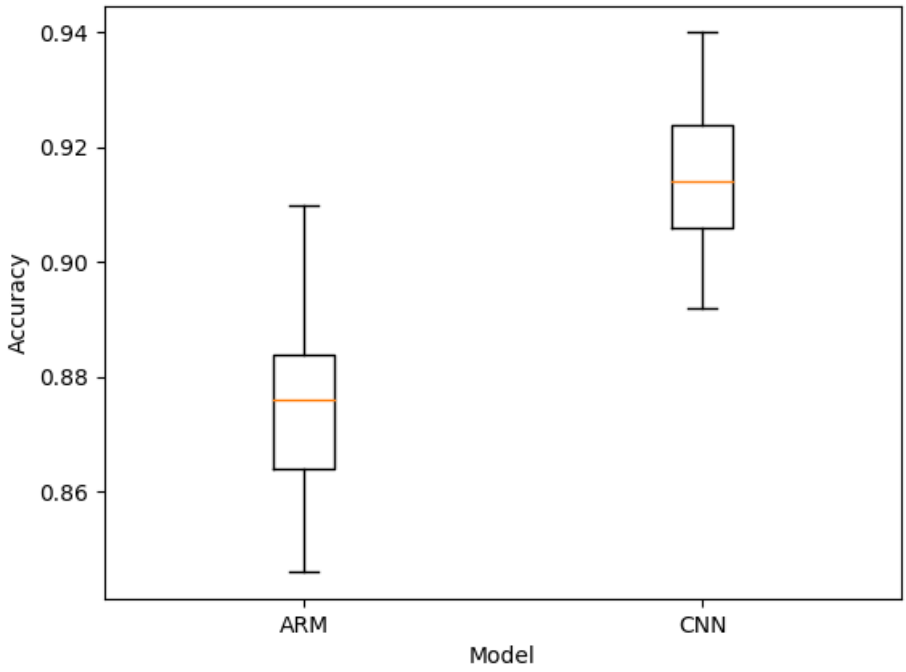

Figure 5.4 Boxplot comparison using 5 training images per class.

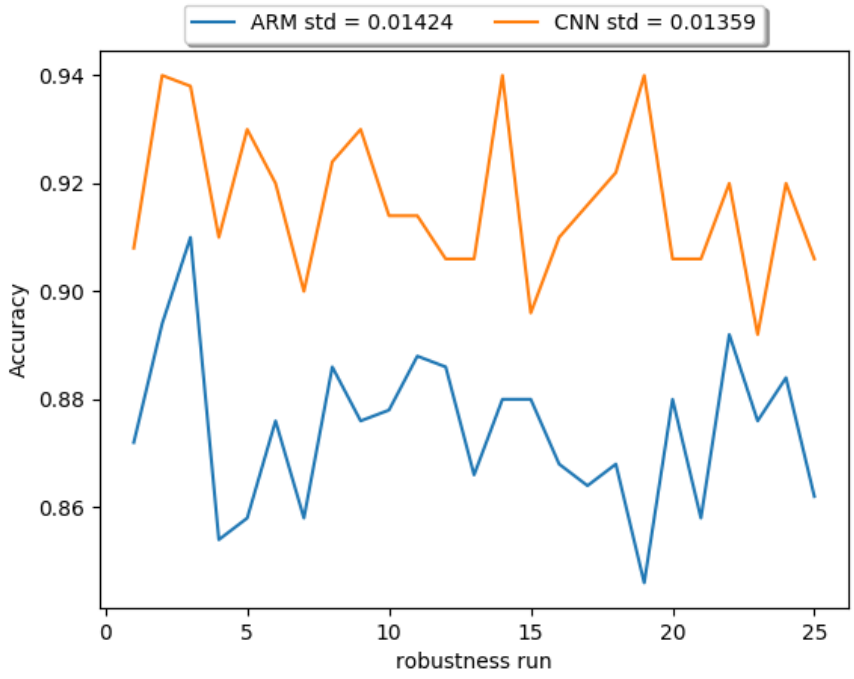

Figure 5.5 Performance comparison using 30 training images per class.

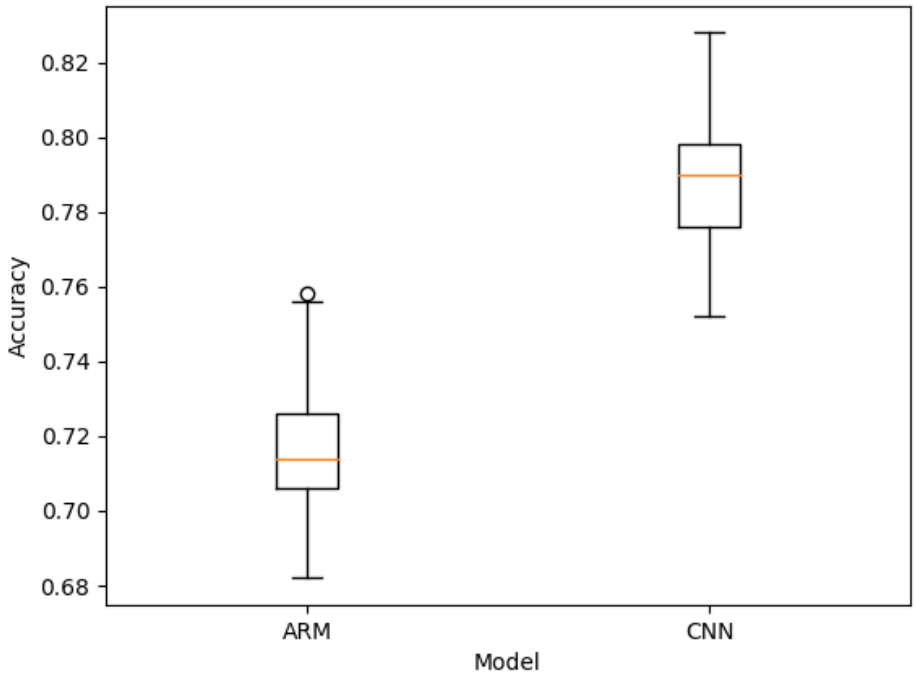

Figure 5.6 Boxplot comparison using 30 training images per class.

The robustness tests are executed among 25 batches of testing data. Each batch contains 500 randomly pooled testing images, and there are 50 images selected for each class. Such testing pools are designed to ensure the model's robustness can be fully tested by new images that are never encountered. Figure 5.3 and 5.4 are the performance plot and boxplot for the ARM and CNN, which are trained given five training data per class. As the performance plot shows, both models fluctuate heavily, and the boxplot indicates that CNN has slightly better robustness than ARM. Both interquartile range (IQR) and min-max range of CNN have less range variation than ARM. However, as more training data is given, such as 30 training data per class, CNN shows similar and even better robustness than ARM, shown in Figures 5.5 and 5.6. Both ARM and CNN's performance plots have similar fluctuation, and the boxplot of CNN reveals that it has a smaller IQR than CNN. Although there is an outliner for ARM, it is closed to the min-max range boundary, and both ARM and CNN's min-max range are about the same length.

Again, since ARM is built purely based on abstractive representations without implementing any optimization processes, logically, ARM is expected to be much less robust than CNN. However, the robustness tests suggest an opposite result. ARM shares similar and even slightly better robustness than CNN. It is concrete evidence that the abstractive representation approach without any optimizations already contains relatively enough information to make stable predictions.

CHAPTER 6: Conclusion

6.1 Findings

From the previous three experiments, it is clear that ARM's overall performance is acceptable as a non-data-driven approach. Although its prediction accuracy is not as good as CNN, ARM shows a similar accuracy rate as CNN. Also, ARM has above 80% accuracy on predicting all ten digits given 30 training data per class, which indicates its ability to handle patterns with different complexities. The robustness tests prove the ARM shares the similar stability as CNN if it is not better.

6.2 Contributions

The most significant advantage of ARM compared to CNN is the explainability. A level-1 pattern binary detector is illustrated in Figure 6.1 to show how the rules are extracted from the detector. A cell value of "1" in binary detector indicates a rule is satisfied, and a cell value of "0" means a rule is not satisfied. The cell's index in the form of (*A*, *B*, *C*) should be converted into a form of (decimal-*A*, binary-*B*, binary-*C*) to illustrate a rule (lvl-0 pattern-*B*, lvl-1 pattern-*C*) of class *A*.

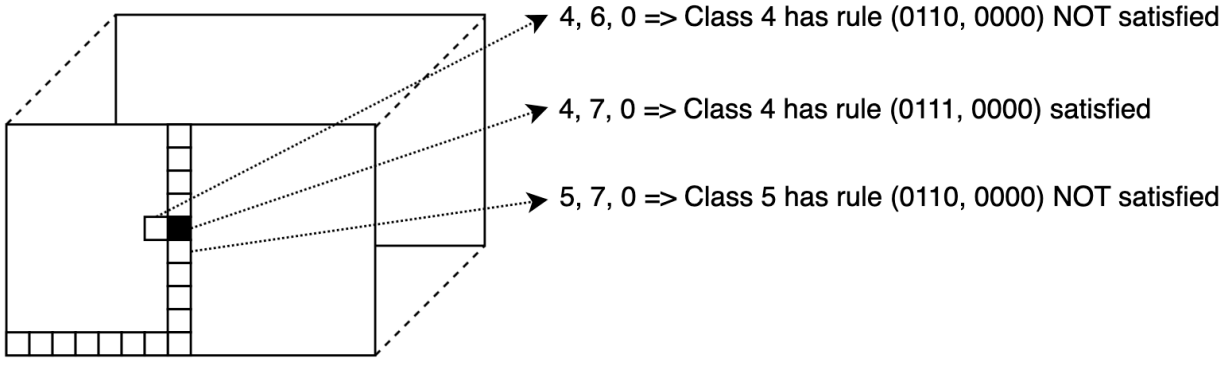

Figure 6.1 Level-1 Pattern Binary Detector.

Table 6.1 illustrates how to interpret patterns in sequence to extract hidden information from the image. Figure 6.2 illustrates the first record in Table 6.1, where four layout patterns of a certain level are satisfied (1000, 0100, 0010, and 0001). As shown in Figure 6.2, there is confusion about the hidden information that the resulting pattern can be either a "centered" position or a "sparse" position. Such an issue can be resolved by going over the detected patterns in the deeper levels. Also, we can further strengthen our conclusion by combining detected and non-detect patterns as the second record shown in Table 6.1.

Table 6.1 Rule interpretation example.

Pattern Combination	Interpretation
4 layout patterns of a certain level are detected. 1000, 0100, 0010, 0001 (illustrated in Figure 6.2)	Previous level pattern is located around the center of the image
2 layout patterns of a certain level are **detected**. 1100, 0100 4 layout patterns of a certain level are **NOT detected**. 0011, 1000, 0010, 0001	Previous level pattern is located horizontally around the top right corner

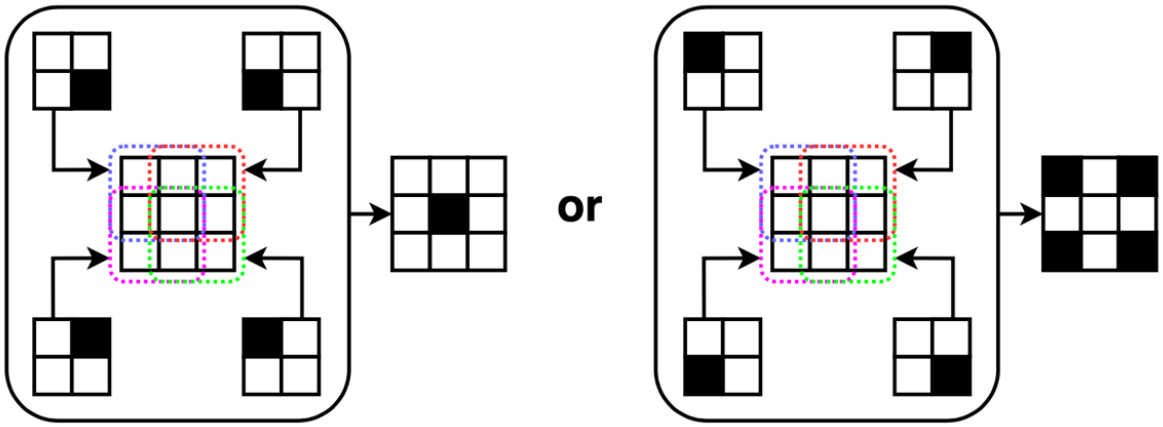

Figure 6.2 Hidden information is extracted from pattern combination.

In Figure 6.3, a workflow illustrates an example of interpreting a satisfied rule (1100, 1100, 1100, 0010) for a partial pattern of number 2. The rules can be interpreted by reading from left to right in a sequence. Each abstractive level is interpreted base on previously accumulated information except the first level (lvl-0). After the rule sequence is reviewed, an abstractive interpretation of a specific pattern is generated by combining all levels' interpretations. However, just like using a single pattern is not as good as using patterns of different levels, using only a single rule is not good enough. The more rules we use in a combination, the more confident our interpretation can be.

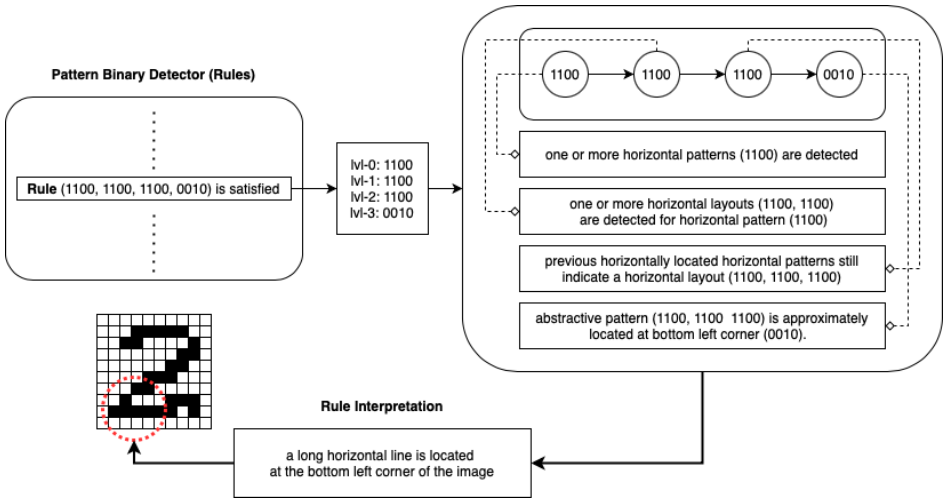

Figure 6.3 Rule interpretation example.

As shown in Figure 6.4, level-3 layout pattern filter "0011" indicates the patterns are located horizontally with empty spaces above, whereas the layout pattern filter "1100" indicates the patterns are located horizontally with empty spaces below. As the layout filter of the final level, considering that "0011" (under) layout is not detected with "1100" (above) layout is detected, we can conclude the pattern is located around the top of the image. More examples of using multiple rules to extract hidden information are shown in Table 6.2.

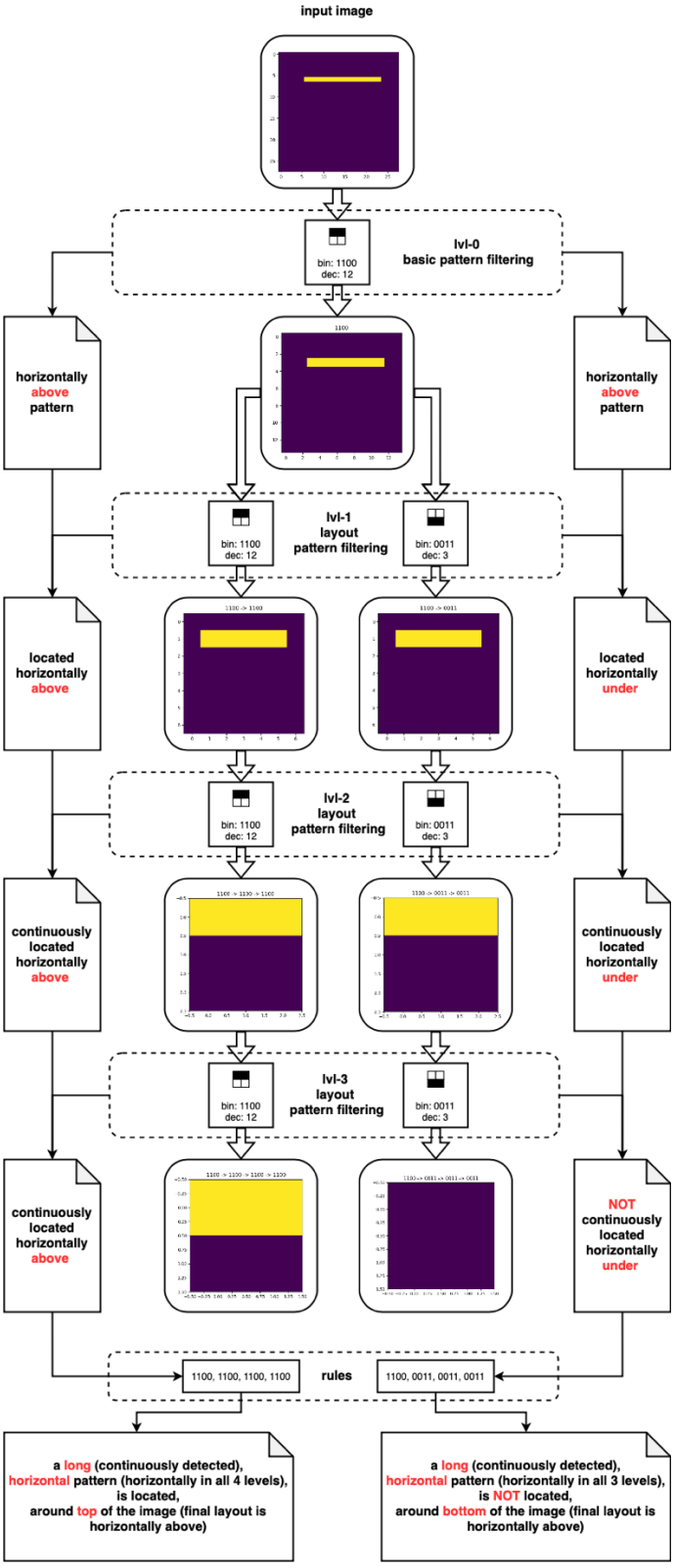

Figure 6.4 Rule interpretation in Combination.

Table 6.2 Rule interpretation example.

Rule	Interpretation
1100 → 1100 → 1100 → 0010 (True)	A relatively **long** horizontal line locates at the **bottom left** corner.
0011 → 0011 → **1100** → [**all lvl-3 patterns**] (False), 0011 → 0011 → **0011** → [**all lvl-3 patterns**] (False), 0011 → 0011 → **1000** → **1000** (True), 0011 → 0011 → **1000** → 0010 (False), 0011 → 0011 → **1000** → 0100 (False), 0011 → 0011 → **1000** → 0001 (False)	A relatively **short** horizontal line locates at the **top left** corner.
0101 → 0110 → 0110 → 0001 (True), 0101 → 0110 → 0110 → 0100 (True), 0101 → 0110 → 0110 → 1000 (False), 0101 → 0110 → 0110 → 0010 (False)	A relatively **long** and sharply **ascending** line locates on the **right** side.

Thanks to the rules, which not only makes it possible to explain the prediction but also makes a "teacher mode" possible for ARM. Before the model processes any training data, if partial or full knowledge about the data is known, then the rules can be predefined to perform the classification without any training processes. It is similar to understanding teachers' lectures instead of spending more time after class to do heavy practice on homework.

6.3 Future Research

Although ARM explains image classification results, its disadvantages can not be ignored. A better data structure is needed for ARM; as the abstractive level proceeds, the computational cost increases exponentially. The current abstractive representation design is based on a single initial basic pattern followed by multiple layout patterns. However, using mixed basic patterns as initial input that is followed by mixed layout patterns can extract even more information from the image. As is mentioned multiple times in previous sections, the lack of optimization affects the performance of ARM in multiple ways. First of all, many useless rules are found to intervene in the decision-making process, and it affects not only the accuracy but also the robustness.

Different rules that share similar meanings need to be identified and grouped, as shown in Figure 6.5, to increase the accuracy and explainability.

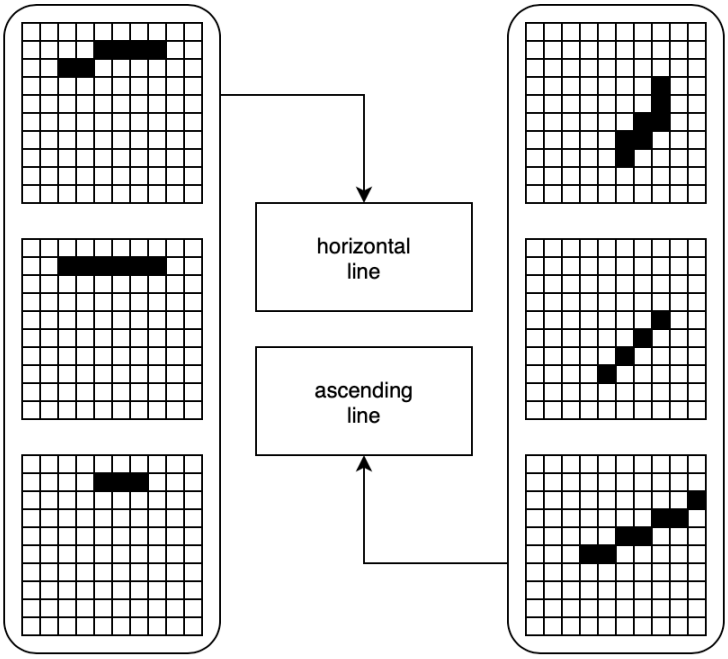

Figure 6.5 Rule clustering example.

In conclusion, ARM is a non-data-driven approach that provides explainability to image classification. Even though it is currently a laboratory approach that is not maturely developed, ARM already shows its potential in research values, provided by its performance and robustness comparison with CNN. More importantly, it directs another possible way of handling image data other than traditional data-driven image processing approaches. ARM is believed to receive better performance for future work with better data structure and optimization processes. From there, developing and improving ARM by applying it to more detailed data such as non-binary images with colors will further test ARM's applicability in real-life problems.

References

[1] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

[2] Wang, S. C. (2003). Artificial neural network. In *Interdisciplinary computing in java programming* (pp. 81-100). Springer, Boston, MA.

[3] Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence, 35*(8), 1798-1828.

[4] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

[5] Stathakis, Dimitris. (2009). How many hidden layers and nodes?. International Journal of Remote Sensing. 30. 2133-2147. 10.1080/01431160802549278.

[6] Borana, J. (2016). Applications of Artificial Intelligence & Associated Technologies.

[7] Teknomo, K. (2006). K-means clustering tutorial. *Medicine*, 100(4), 3.

[8] Liu, B., Xia, Y., & Yu, P. S. (2000, November). Clustering through decision tree construction. In *Proceedings of the ninth international conference on Information and knowledge management* (pp. 20-29).

[9] Muda, Z., Yassin, W., Sulaiman, M. N., & Udzir, N. I. (2011). A K-Means and Naive Bayes learning approach for better intrusion detection. *Information technology journal, 10*(3), 648-655.

[10] O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

[11] LeCun, Y. & Cortes, C. (2010). MNIST handwritten digit database.