9-29-2020

Approximate Pattern Matching Using Hierarchical Graph Construction and Sparse Distributed Representation

Aakanksha Mathuria
Portland State University

Approximate Pattern Matching using Hierarchical Graph Construction and Sparse Distributed Representation

by

Aakanksha Mathuria

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Dan Hammerstrom, Chair
Christof Teuscher
Nirupama Bulusu

Portland State University
2020

Abstract

With recent developments in deep networks, there have been significant advances in visual object detection and recognition. However, some of these networks are still easily fooled/hacked and have shown ”bag of features” kinds of failures. Some of this is due to the fact that even deep networks make only marginal use of the complex structure that exists in real-world images. Primate visual systems appear to capture the structure in images, but how?

In the research presented here, we are studying approaches for robust pattern matching using static, 2D Blocks World images based on graphical representations of the various components of an image. Such higher-order information represents the ”structure” or ”shape” of the visual object. This research led to a technique for representing an object's structural information in a Sparse Distributed Representation (SDR) loosely based on the kinds of cortical circuits found in primate visual systems.

We apply probabilistic graph isomorphism and subgraph isomorphism to our 2D Blocks World images and achieve $O(1)$ and $O(n^k)$ complexity for an approximate match. The image labeled graph is created using OpenCV to find the object contours and objects' labels and a fixed radius nearest neighbor algorithm to build the edges between the objects. Pattern matching is done using the properties of SDRs. Next, we use SVM to learn and distinguish images. SVM partitions the vec-

tor space where classification accuracy on noisy images gives us an assessment of how much information the SDR is capturing.

To My Family.

Acknowledgements

First of all, I forward my greatest regards to God.

I would like to thank my advisor Dr. Dan Hammerstrom for his helpful support and guidance in research throughout my MS degree. His insightful views and the countless number of research ideas inspired me in this interdisciplinary area of research. He encouraged me and gave valuable assistance for the completion of my thesis.

I am deeply and forever indebted to my family. My parents Bharat Mathuria and Santosh Mathuria, sisters Anjali and Amrita, and brother Aaryan always supported me and gave me strength to face and fight any kind of difficult situations. Without them the completion of the thesis work was in no way possible.

I would also like to thank the Center for Brain Inspired Computing (C-BRIC). This work was supported in part by the C-BRIC, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, SRC Award Number 2018-JU-2777.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Part of the work has been published in an International Conference on Neuromorphic Systems (ICONS) paper in 2019 [1]. Some passages included in the thesis have been taken from this paper.

With the recent advances in deep networks, there has been significant progress in visual object detection and recognition. However, some of these networks have shown "bag of features" failures [2] similar to the other traditional object recognition techniques such as HOG (histogram of oriented gradients) [3], SIFT (Scale-invariant feature transform) [4], [5] and special envelope [6]. Bag of features is a collection of features with no order, structure, or spatial relationship [7] like, we have all the features of a bicycle in an image but not in the right structure. Deep networks make only marginal use of the complex structure that exists in real-world images, even after training on large numbers of images. None of these techniques actually captures the spatial relationships of the low level or high-level features, which biological networks appear to do [8], [9]. There has been some previous work trying to understand shapes and objects [10].

Efficient graph representations capture the higher-order information content of the objects and provide algorithmic benefits when recognizing complex images [11], [12]. Such higher-order information represents the "structure" of the visual

objects. Also, an important difference in the work described here is that we are using a non-standard representation of the graphical data based on sparse distributed representations (SDR). SDRs are large binary vectors with a few active bits. We are representing structure by a graph and then graph via SDRs. This helps us in reducing complexity.

Neuromorphic techniques such as Sparse distributed representations (SDR) of data, shapes, and graphs can play an important role in complex image processing. The use of sparse representations of data is motivated by a) the abundance of visual data b) the abundance of features in real-life images and c) the ability of sparse representations to provide speed up via unique properties (e.g. union) of the representations. An SDR encodes any type of data into a binary vector which consists mostly 0's with a few 1's. SDR is very memory efficient, as only a few bits would have to be stored in the memory as the indices of the active bits [11]. SDRs are the result of various research efforts into understanding the operation of cortical circuits [13], [14].

In the research described here, we are exploring new ways to represent images as hierarchical graphs to preserve the relative connectivity information among the objects and perform pattern matching using graph isomorphism. The hierarchy allows us to reuse the low-level information into higher-levels. The graph of an image uses objects as the nodes. It contains the spatial information (connectedness, adjacency) of the objects in the image. The connections can represent the Euclidean distance between the nodes. We formulate SDRs for all the nodes in the graph using their attribute information such as the number of edges, their sizes, connectivity, and attributes of their neighbors. Then we use Euclidean distance criteria to represent the hierarchy in the graph, which can be used for efficient pattern matching.

An example of a hierarchical graph construction for an image containing multiple individual components can be used with three levels. For the first level, we can consider small body parts such as nose, mouth, eyes, etc. as nodes for a graph representing the face of a person. Each of these small body parts can be represented by SDRs with their attributes. Similarly, graphs of other large body parts, such as hands, legs, etc., can be defined. With the properties of SDRs, such as union, one can define SDRs for the entire graph, in this case of large body parts such as hands, legs, and face, etc. As a second-level hierarchy, the graph can be constructed of these large body parts as nodes and connectivity between them and the graph representing an entire individual. Again, an SDR of this entire graph can be obtained by performing a union over the SDRs of the individual nodes. To construct the graph of the entire image with different individual components, the SDRs of each person can be considered as a node of the graph. This type of representation promises an efficient pattern-matching algorithm when implemented using graph isomorphism. Figure 1.1 shows ways an object can be seen as a group of blocks.

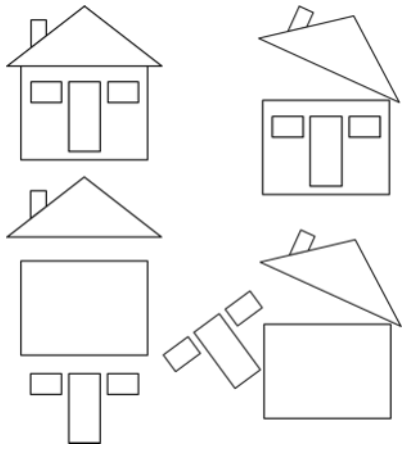
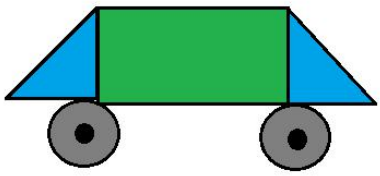


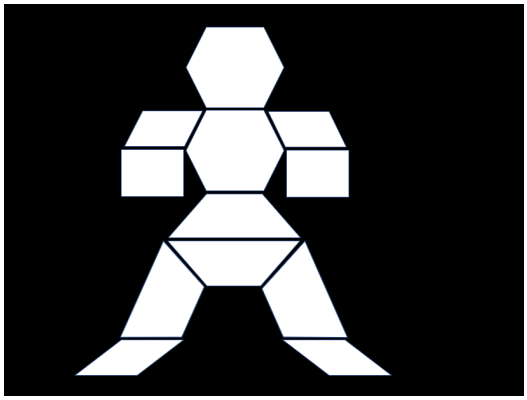
Figure 1.1: Object seen as a group of blocks.

To demonstrate these ideas, assume simple objects, e.g., rectangles and trian-

gles, from a 2D blocks world. These are recognized using traditional algorithms (OpenCV). We then create graphs of these objects to allow the efficient recognition of more complex objects, built from the simple objects. Figures 1.2 and 1.1 show how real-world objects can be broken into simple blocks that can be easily and effectively represented using SDRs.



(a) Simple Blocks-World image



(b) Complex Blocks-World image

Figure 1.2: Real world images seen as Blocks-World, similar to a (a) vehicle, and (b) human

In this work, we use probabilistic graph isomorphism and sub-graph isomorphism to perform efficient approximate pattern matching in images. The optimal match is an NP-hard problem. However, with the help of SDR properties, we can perform graph matching in $O(1)$ time and further choose k nodes sub-graph out of the main graph of n nodes in $O(n^k)$ and do the matching in $O(1)$. k is the number of nodes in the other graph ($n > k$). By combining the SDRs and graphs, we can perform pattern matching, which leverages structural information in an efficient manner. Here, pattern matching is based on the shape of the object, not so much on individual features.

Next, to evaluate and assess how much information the SDR is capturing of shapes and structure in an image, we use the Support Vector Machine (SVM). Us-

ing image graphs and SDRs, we train the SVM model. SVM classifies the new test image SDR to the already leaned classes. Thus it helps us in recalling the original image from a noisy or incomplete version of it. Our definition of noise encompasses not only traditional measures of noisy, such as speckled images, partial occlusion, but also "shape" noise, which involves missing, extra, or poorly placed components.

1.1 Contributions

The contributions of this thesis include:

1. We create a hierarchical graph representation to capture the structural information of an image.

2. We implement Sparse Distributed Representations (SDR) for the hierarchies of a graph, which leverages algorithmic parallelism and makes computation faster and more power-efficient.

3. We demonstrate the approximate graph matching in $O(1)$ and by choosing k nodes' sub-graph out of n nodes' big graph in $O(n^k)$, sub-graph matching in $O(1)$ instead of solving in non-polynomial times with the help of SDR properties.

4. We use SVM to partition the SDR vector space, which gives us 94% accuracy for shape and traditional image noise, and 98% accuracy with only the addition of traditional noise for the retrieval of original images from noisy versions of it.

5. Our encoding, currently with blocks world objects, is a reasonably robust encoding of the structure of an object. This allows us to do a better job of rec-

ognizing an object by its shape. In addition, we have expanded the definition of image noise to include "shape" noise.

Our method allows us to capture structural information in images for doing pattern matching and uses very little data.

The work described in this thesis builds on the work presented in my M.Tech thesis at Indian Institute of Information Technology Allahabad (IIITA), India [15]. The work was done in the guidance of Dr. Dan Hammerstrom (PSU) and Dr. Uma Shanker Tiwary (IIITA). A basic idea of combining graph analytic and SDRs to do pattern matching was developed in the IIITA thesis. For this, a preliminary implementation using simple block images was done.

In this thesis, we deal with much more complex figures generated using the BlocksWorld tool. The SDR encoding is further re-implemented in Python and improved using flocet codes to evaluate the similarity of vectors. Attribute height-width ratio is introduced in SDR encoding for node and their neighbors to ensure scale invariance. Attribute orientation angle for nodes and their neighbors is also incorporated to improve the capturing of structure. For example, in a face, the correct position and the correct angle of all the parts, both are important for identifying the face. In the IIITA thesis, we introduced the algorithms for doing graph isomorphism and sub-graph isomorphism separately. Here, we propose and implement an algorithm given two graph SDRs to check whether two graphs are isomorphic and if not, whether they are sub-graph isomorphic. The sub-graph isomorphism algorithm is improved by leveraging the hierarchical image graph. We only check for the sub-graphs which respect the hierarchy which involves small, local neighborhood graphs and not an entire image.

In addition to the improvement of previous work, we assess the ability of SDRs to learn the images and their structures. For this, we train the image SDRs and then try to retrieve the shape and image information from a noisy or incomplete version of it. We introduce the shape noise notion for the Blocks-World images. The retrieval accuracy is observed for two types of noise, traditional image noise (partially blocked components, speckle or colored patches on the components), and shape noise (extra, missing components).

We first trained the object SDRs of an image using a simple auto-associative memory model. Then we test the model in three ways, (a) with the same SDR vectors without adding noise, (b) with the same SDR vectors with added bit noise, and (c) with noisy object SDRs. Associative memory works well when the number of training vectors is less, and the images are less complex. The auto-associative memory model did not work when we train the image SDRs. Image SDRs are much more complex as they store the information of all the individual objects and their relationship with others.

Therefore, we moved to the Support Vector Machine (SVM) that allows for more complex and non-linear partition. While training, SVM partitions the vector space for the training image SDRs. Each image is considered a different class. When inputting an image, the SVM model classifies the image and recalls the closest matched image SDR. We train the SVM model on 10 image SDRs and test it on 180 noisy image SDRs (shape and traditional noise). For this, we achieve the retrieval accuracy of 94%.

To further improve the retrieval accuracy, we worked with Dr. Rod Rinkus on Sparsey. Sparsey is a multi-layered network with a property that similar inputs are mapped to more similar codes (SISC). Sparsey maps the SDRs to more distributed sparsified vectors. The vectors are now more distinct from each other

which should help in achieving better retrieval accuracy. We continue to work on Sparsey for the classification of SDRs and also for using the sparsified vectors with the SVM model.

1.2 Thesis Organization

Here, we describe the organization of this thesis. First, we discuss the steps to create a hierarchical image graph (chapters 2, and 3) and the encoding with which we can store the structural and connectivity information of an image (chapter 4). Then we discuss the approach we can use image graph and their SDRs to do pattern matching (chapter 5). In chapter 6, we show another application of an image graph and SDR in retrieving the data (image in our case) from a noisy version of it. And, chapter 7 summarizes the thesis work and discusses some future directions.

The approaches and steps described in chapters 2, 3, and 4 are applied to both our image graph + SDR applications (pattern matching, and retrieval of information). The chapters are organized into four sections (a) introduction and Motivation (b) related work, (c) approach, and (d) results. In chapter 2, we describe the process of object detection and the features extracted using OpenCV. In chapter 3 we describe how to form a hierarchical graph from the detected objects as nodes with a fixed-radius nearest neighbors algorithm. Then in chapter 4, we discuss the possibility of representing graphs in SDRs and leveraging the massive parallelism, for example, in massively parallel associative memories, that is enabled by SDRs.

In chapter 5, we present an algorithm using SDRs for exact and approximate matching using the generated image graph SDRs. Figure 1.3 shows the data flow of approximate pattern matching using the combination of graphs and SDRs. In chapter 6, we show how SVM partitions the image graphs' SDR space to classify

the test image. Here, the test image is a corrupted version of an already learned image. SVM helps us in recovering the originally stored/learned image from a corrupted/noisy version of it.

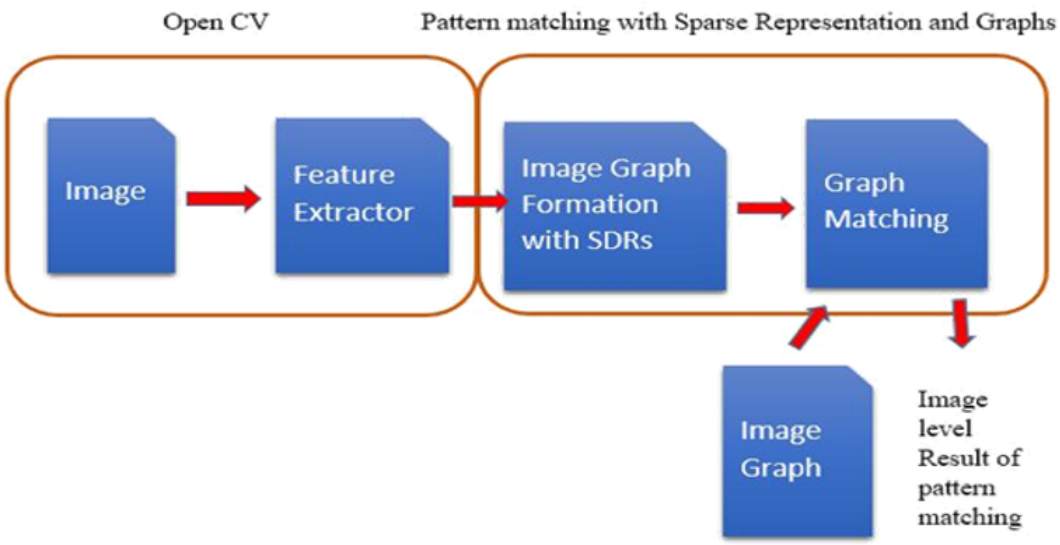


Figure 1.3: Data flow pipeline for pattern matching [1]. Input is a Blocks-World image. OpenCV is used to detect the object contours and to extract the features of the contours. Using the contours and their features, we generate image graphs and encode object attributes into SDRs. Pattern matching of two images is done using their SDRs.

Chapter 2

Object Detection

2.1 Introduction and Motivation

Object detection has been a very important task of any computer vision application. Humans can easily detect an object irrespective of the scene or position they are in. It is critical in many applications such as optical character recognition, surveillance, self driving cars, and medical imaging. Given an image or a Region of Interest (ROI), the goal of object detection is to find the locations of objects in the image and to classify them. Object detection can be used to identify the number of objects present in the image. We can also track their precise locations, all while labeling them. Figure 2.1 shows the objects detected in a scene using bounding boxes.

In this chapter, We take a blocks-world image and use OpenCV functions to get the image parts. These parts belong to composite objects in the image. OpenCV finds the contours in the image as well as their features such as centroid, minimum area bounding box, and perimeter. The features are used to calculate the attributes which will be encoded into SDRs to store the image information.

Figure 2.1: Objects detected in a scene using bounding boxes.

2.2 Related work

Many researchers and scientists have been working in this field for many decades now. There are many applications from face detection and pedestrian detection to image and video retrieval. Currently, object detection is an integral part of many common areas like video surveillance, image captioning, video summarization etc. There are many techniques to detect the objects in an image. Some techniques use feature extractors, like SIFT [4], [5], and HOG [6]. Some use bounding boxes and contour detection. Deep learning techniques also efficiently localize and classify the objects in an image, such as R-CNN [16], Faster R-CNN [17], SSD [18], and YOLO [19]. Figure 2.2 shows low-level features detected from SIFT in an image and figure 2.3 illustrate the YOLO model on an image.

Figure 2.2: Low level features detected in an image using SIFT [20].

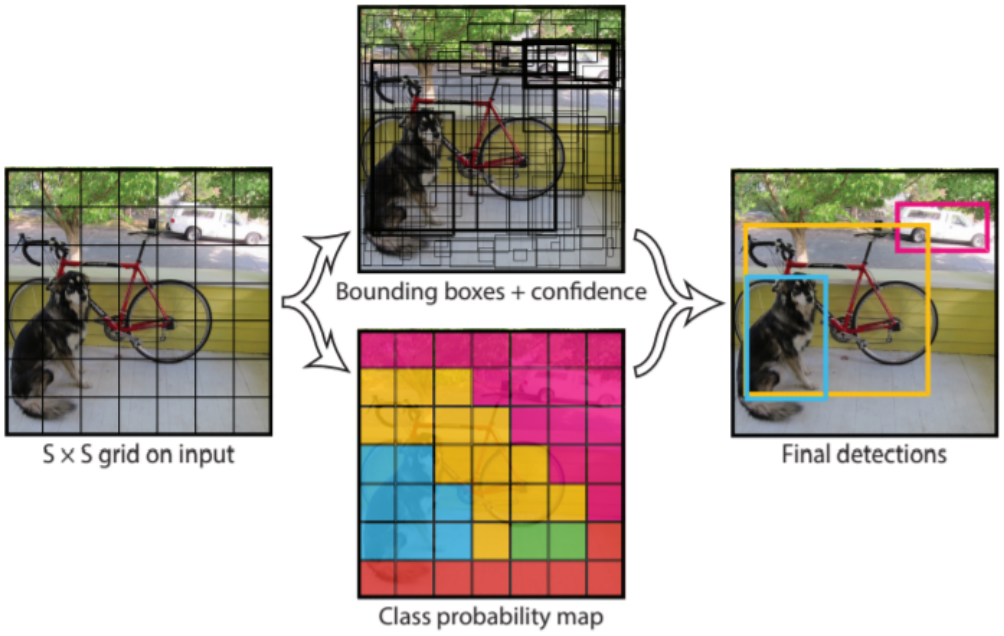


Figure 2.3: YOLO model for object detection [19].

2.3 Approach

Traditional computer vision uses low-level features from the images. In this thesis, we use high-level features obtained from OpenCV. The objective is to capture the

connectivity of components in an object. In this thesis, instead of using a bounding box, we use shape contours to locate the components of the image. OpenCV gives us the contours of the components as well as the features of the contour shapes. We utilize these features to further determine the shape attributes. After calculating the attributes, we would encode them into a Sparse Distributed Representation (SDR) for further processing.

Contour tracing/extraction is an important technique in image processing for shape analysis and object detection. A contour is defined as the curve joining all the points along the boundary of a shape having the same intensity or color. For the shape contours, we use OpenCV's *findContours()* method. The function is applied on a binary image with shapes in white and background in black. To achieve better accuracy, an image is pre-processed before applying the function. Images used in this thesis are with black objects and white background. Hence, the image is converted using bitwise NOT operation. The *findContours()* method has three essential parameters, source image, contour retrieval mode and, contour approximation method. There are five types of retrieval mode. We use RETR_EXTERNAL mode to retrieve only the extreme outer contours. For the approximation method, we choose the CHAIN_APPROX_SIMPLE method to extract only the end points of horizontal, vertical, and diagonal segments by removing all the redundant points, thereby saving memory. The outputs of *findContours()* are a modified image, contours (list of all the contours in the image) and, hierarchy. Each contour has (x, y) coordinates of boundary points. Hierarchy contains the information about the image topology. Figure 2.4 represents the computed contour from *findContours()* method.

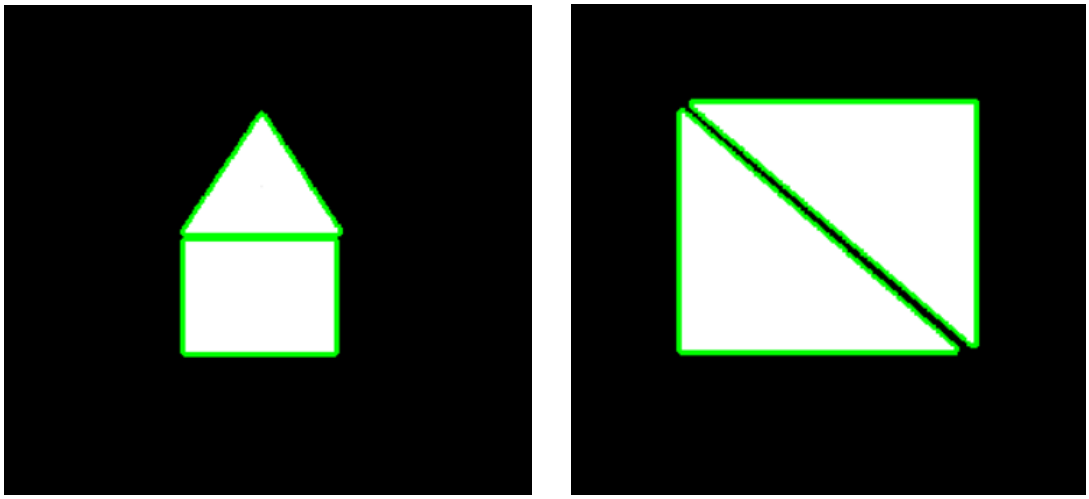


Figure 2.4: Shape contours detected from OpenCV's *findContours() function*.

2.3.1 Attribute Computation

Using the contour list, we compute the features of the shapes, such as centroid, area, size of the minimum bounding box, angle of rotation, and perimeter using OpenCV's functions. After that, based on our application and the images used, 2D Blocks world images (section 2.3.2), we define the attributes of the shapes as (a) Number of edges (b) Height-width ratio (c) Orientation angle (d) Connectivity (number of connected neighbors), and (e) Relative position (angle with the neighbor) for further processing.

To determine the contour centroids, we use image moments. An image moment is a particular weighted average of the image pixel's intensities. For an image intensity function $I(x, y)$, raw image moments M_{ij} are calculated from equation 2.1. Centroids can be derived using these raw image moments as shown in equation 2.2. To calculate image moments, we use OpenCV's moments() function which

takes a contour and outputs moments.

$$M_{ij} = \sum_i \sum_j x^i \, y^j \, I(x, y) \tag{2.1}$$

$$C_x = \frac{M_{10}}{M_{00}}, \quad C_y = \frac{M_{01}}{M_{00}} \tag{2.2}$$

For area and perimeter, we use OpenCV functions *contourArea()* and *arcLength()* respectively. Area can also be obtained by moment $M['m00']$. Size of the minimum area bounding box and angle of rotation are calculated using function *minAreaRect()*. All of these functions take contour as inputs. Figure 2.5 shows a shape's bounding boxes. The red box is a rotated one from function *minAreaRect()* which also gives us the angle of rotation.

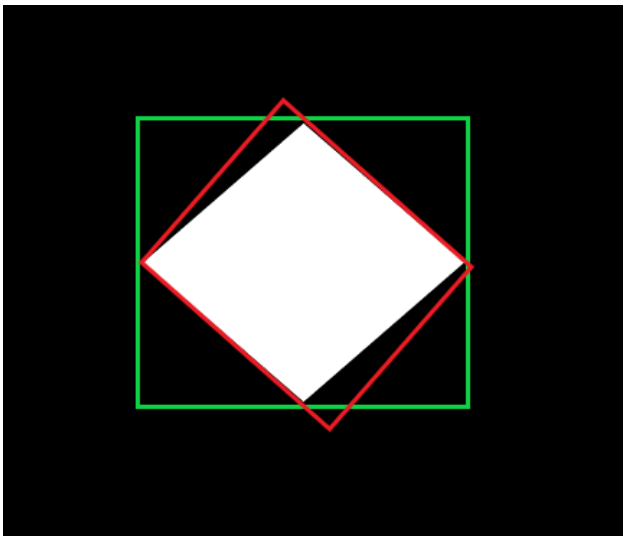


Figure 2.5: Green rectangle represents normal bounding box and red rectangle box represents minimum area bounding box.

All these features are further used to calculate the shape attributes for SDR. Figure 2.6 represents a shape in the image with the computed attributes. How each attribute is computed is shown below:

1. Number of edges is calculated from the perimeter of the shape contour from equations 2.3 to 2.5. Equation 2.4 approximates the shape with the specified precision. *epsilon* is the accuracy parameter. To get the correct output, a good epsilon needs to be selected, x% of perimeter.

$$perimeter = \text{cv2.arcLength}(contour) \tag{2.3}$$

$$approx = \text{cv2.approxPolyDP}(contour, epsilon) \tag{2.4}$$

$$\text{numberOfSides} = len(approx) \tag{2.5}$$

2. As discussed above, function *minAreaRect()* gives us the size of the bounding box, width and height, from this we calculate the height-width ratio. The function also gives us the angle of the bounding box which is the orientation angle of the shape. Note, one result of using these functions means that the resulting parameters are independent of position and scale.

$$(center,\ size,\ angle\_rotation) = cv2.minAreaRect(contour) \tag{2.6}$$

$$HW\_ratio = \frac{size[1]}{size[0]}, \quad Orientation\_angle = angle\_rotation \tag{2.7}$$

3. Connectivity and relative position are calculated after constructing the image graph. Once we have the graph, the number of connected neighbors for every shape is determined by the adjacency list. Relative position is the angle with the neighbor, which we calculate using the centers of both the shapes.

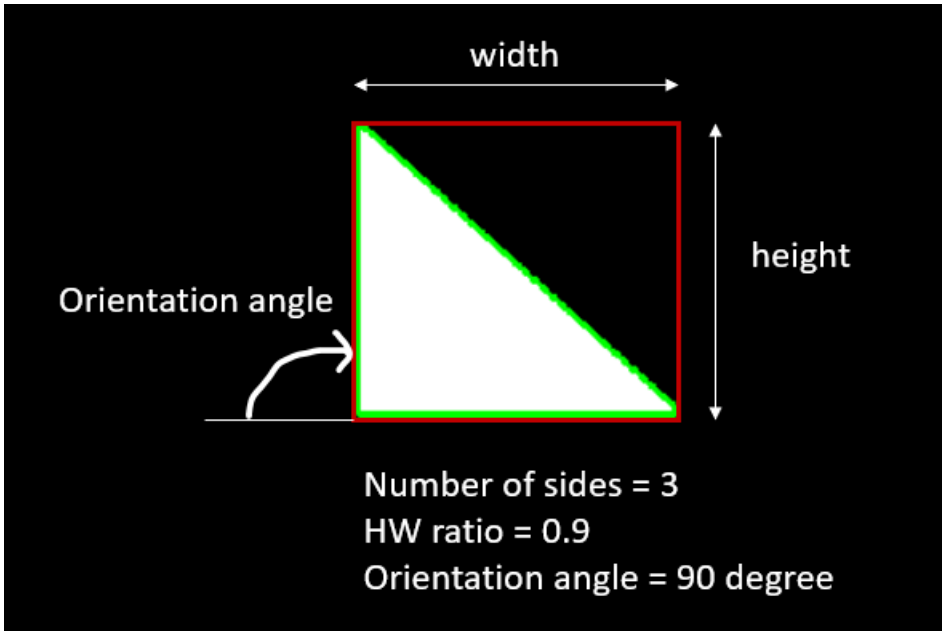


Figure 2.6: Attributes of a triangle. Red rectangle represents the minimum area bounding box. Height, width, and orientation angle of the bounding box are the height, width, and orientation angle of triangle.

2.3.2 Image generation tool

To generate simple blocks world images, we use the BlocksWorld tool [21]. We can build several blocks world images from this. A blocks world image is an image where the objects are made of a few blocks (shapes). The number of blocks in each object is random. First, it generates a central object around a center of randomly chosen vertices. Then it randomly takes one of the available edges and creates another shape along with it. This continues until we get the required number of blocks in the object. It can generate 2.5D images since it is planar but allows objects to occlude each other.

With the help of BlocksWorld, we can create various objects in an image consisting of several blocks. For the work described in this thesis, we require that the blocks are not overlapped. While using the tool, we have set an offset between

each block so that the blocks don't overlap. For every block created, BlocksWorld checks whether the block obstruct any of the existing ones. Figure 2.7 shows some example images from the BlocksWorld tool used in this work.

Figure 2.7: BlocksWorld images generated with BlocksWorld tool [21].

Chapter 3

Hierarchical Graph Creation

3.1 Introduction and Motivation

In this chapter, to capture the image structure, we will generate a hierarchical image graph and will consider the OpenCV's detected parts' centers as nodes. Graphs are useful when one wants to represent the connectivity or structure of objects. A hierarchical graph is useful to reuse the low-level features to further combine them at higher levels. Figure 3.1 shows a giraffe in the jungle on the left side and a structure of its features' connections on the right side. Applications, such as document processing, scene processing, image retrieval [22]–[25] and video summarization [26] could benefit from such connectivity information. However, due to the complexity of working with graphs, traditional Computer Vision techniques mostly use a "bag of features" [27], [28] approach and so are missing information on object structure.

Humans factor shape into object recognition consequently features being in the wrong position degrade recognition accuracy. Imagine two images of bicycles, one being with the right position and orientations and other being with only the right components and wrong locations. When we classify this image using a "bag of features" approach both images will be classified as bicycles, but the second image is not the correct form of a cycle. Being able to utilize such structure or connectivity

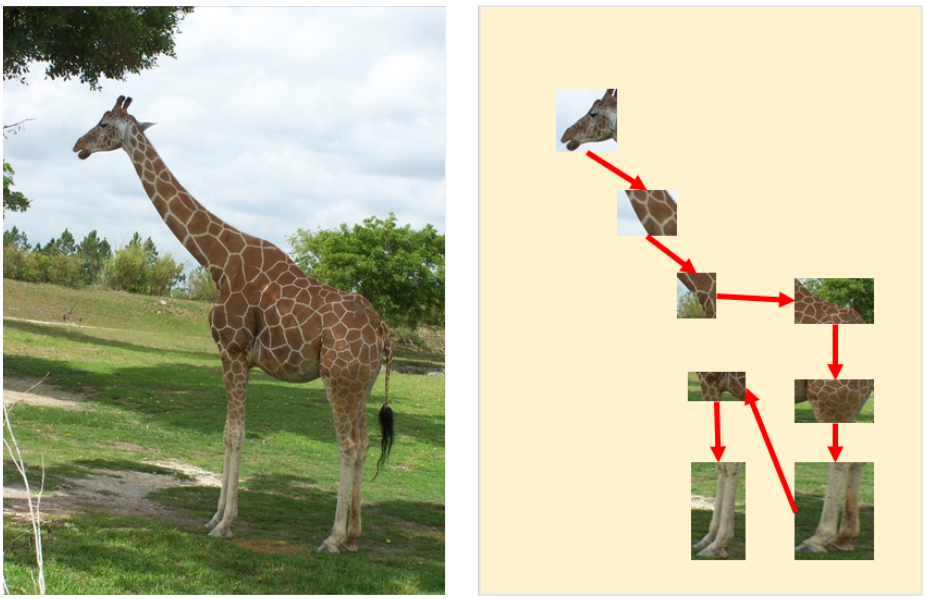


Figure 3.1: Features of giraffe called out to show the structure.

information will be of significant value in image understanding. One approach to representing structure in deep networks is the development of Capsules [29] by Geoffrey Hinton and his group. Capsules take advantage of the fact that spatial relationships can be modeled by matrix multiplications.

To capture the connectivity between the objects and to reduce the complexity and memory usage, we generate a hierarchical image graph. And, there is no question that cerebral neo-cortex processes data and represents it in a hierarchies [30]–[32]. For this, we use fixed-radius nearest neighbors algorithm with a radius r. The fixed-radius near neighbor problem is a variation of nearest neighbor search. Nearest neighbor search is the problem of finding the closest point n to a given query point q among a set of points S. The proximity between the points can be defined from a distance matrix. The goal of the variant, fixed-radius nearest neighbors problem is to find all points N in the Euclidean space of points S within a given fixed distance $r > 0$ from a specified point q [33]. Figure 3.2 shows an

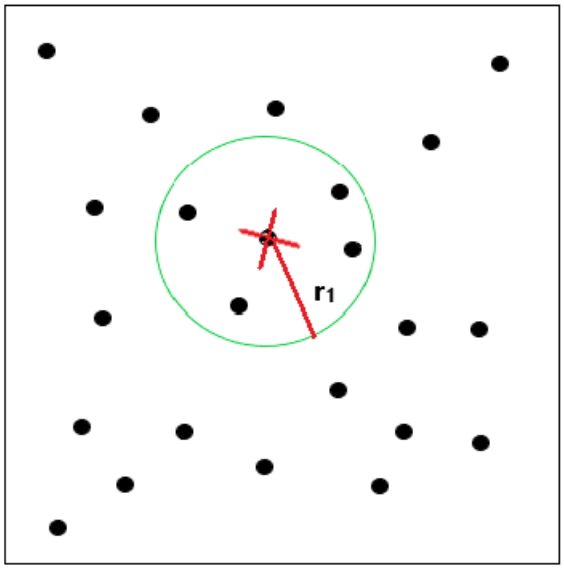


Figure 3.2: Example of fixed-radius nearest neighbor problem. Area inside green circle is the euclidean space for radius r_1.

example fixed-radius nearest neighbor solution. There are two points for which the solution is found separately with different radii. Red points in two circles belong to their respective Euclidean spaces, hence the solution.

3.2 Related work

Traditional computer vision techniques typically do not capture the locality and connectivity of objects [34]. Traditional (pre-deep network) systems find complex information associated with each feature, using feature detection algorithms, such as SIFT [35]. Then the discovered features are matched somewhat independently to a set of features associated with each object, which has been termed a "bag of features" approach. Determining the presence of an object solely by its features gives unsatisfactory results [34], [36]–[38].

3.3 Approach

In this thesis, we leverage the properties of hierarchical graphs for pattern matching. In a hierarchy, we store the lowest level of information for reuse when combined in novel ways at higher levels for complex information [13]. Hierarchies reduce training time and memory usage. We represent the images using hierarchical graphs to get better accuracy with simple and complex images. Graphs help in keeping the connectivity information intact while processing and hierarchy help in sharing the information among different levels. It is clear that biological vision, at least in mammals, takes advantage of the geometric relationships of the features with each other, which we refer to as the "structure" of the object. It appears that the visual cortex at the lowest level of the processing hierarchy stores information about tiny sections of the visual field such as edges and corners [13]. These low-level patterns are recombined at higher levels for more complex components.

Here, we assume that all the detected shape contours from OpenCV in the image are parts of much more complex objects. Therefore, in our 2D blocks world images, we assume three levels of hierarchy in an image graph. The level 1 graph is among the detected parts which are treated as nodes. All the connected components from level 1 are treated as separate objects in a level 2 graph. These objects are made of the parts from level 1 based on their proximity to each other. Level 3 is the image itself. The number of levels can vary based on the complexity of the image and the application. The graph is constructed using the fixed-radius nearest neighbors algorithm. For this, we use the `NearestNeighbors()` function of the sklearn.neighbors module with the radius parameter on centers as shown in equation 3.1. r_1 is the radius and centroids are the image parts' centers. `NearestNeighbors()` is the unsupervised learner for implementing neighbor

searches and `radius_neighbors` find the neighbors within a given radius of a point or points [39]. *indices* arrays are the indices of neighbors and *distances* array represents the distances to each neighbor point.

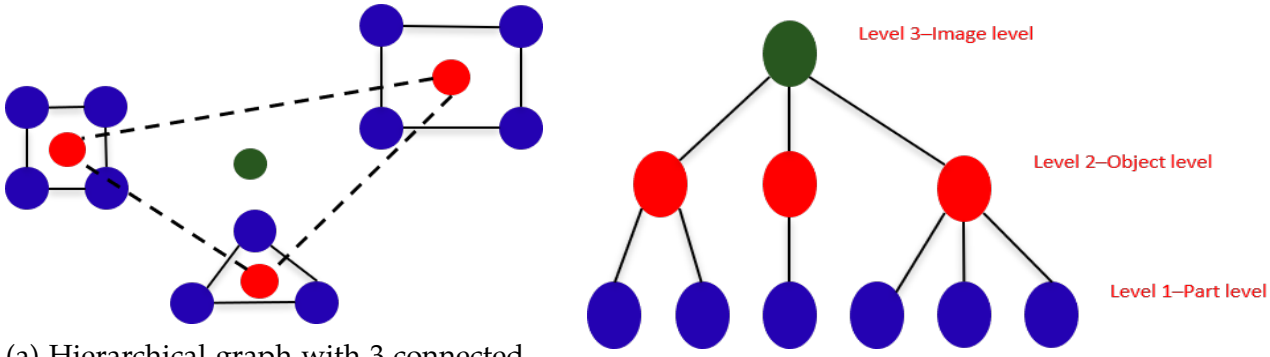
$$\begin{aligned} neighbors &= NearestNeighbors(radius = r_1).fit(centroids) \\ distances, indices &= neighbors.radius\_neighbors(centroids) \end{aligned} \tag{3.1}$$

The level 1 image graph is built using the fixed-radius nearest neighbors algorithm with radius r_1, parts' centers are considered as nodes. This level shows the spatial relationship between the parts. We calculate the connected components from the level 1 graph. In graph theory, a connected component of an undirected graph is a sub-graph in which any two vertices are connected to each other by more than one path, and which is not connected to additional vertices in the super-graph [40]. These connected components are considered objects in the image and will be treated as nodes for the level 2 graph.

Level 2 of the graph is constructed between these objects by applying the fixed-radius nearest neighbors algorithm with radius r_2 ($r_2 > r_1$) [1] to the new centers. Radius r_1 is a constant which we choose so that all the components belonging to an object are connected whereas for r_2, also a constant, we assume that all the objects would be connected in level 2. r_1 and r_2 are determined such that while scaling certain component or object to an extant the connections in the image graph don't change and the height-width ratio ensures the scale invariance. The new centers are calculated by finding the spatial arithmetic mean of the parts' centers, parts that belong to that specific object. If we need more levels for complex images then they can be created by applying the fixed-radius nearest neighbors algorithm to new calculated centers from the previous level's connected component centers'

[1] r_1 and r_2 are manually set based on image size and complexity.

arithmetic mean. Figure 3.3 shows an example of a graph with three connected components.



(a) Hierarchical graph with 3 connected components in level 1

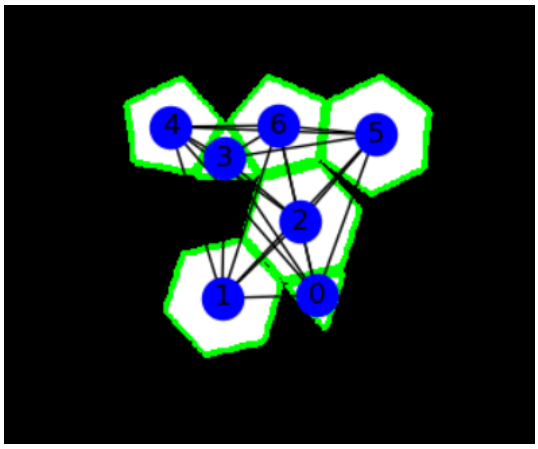
(b) Corresponding graph hierarchy

Figure 3.3: Hierarchical graph demonstration, blue nodes are the parts, red are the objects, and green nodes represent the image. In (a), solid lines belong to level 1 graph and dashed lines belong to level 2 graph [1].

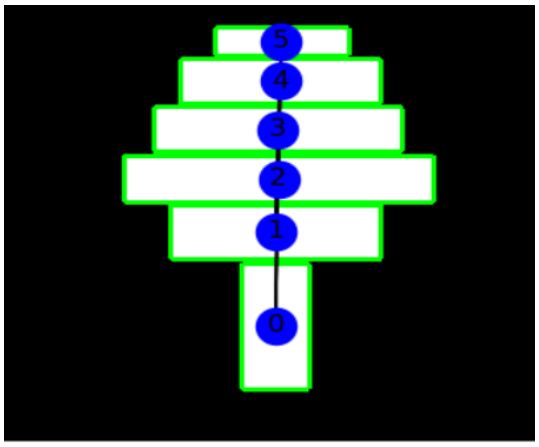
3.4 Results

Here, we show the detected parts and their generated graphs for the blocks world images. In figure 3.4, we have five images and all of them have only one object made of composite parts. These images have only 2 graph levels, one is the part level and the second is the whole object or image level. Here, the graph is only between the object's parts and how they are connected to each other.

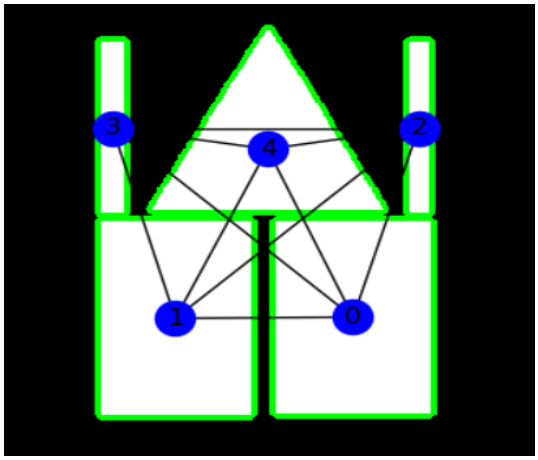
Figure 3.5 has images with multiple objects, made of some parts, far enough to be separate objects. Left side images represent level 1 graphs and right side images represent corresponding level 2 graphs. The lowest level (level1, represented by blue) of the graph represents connectedness between the basic detected parts, which, in turn, make complex objects in the image. The second level (represented by red) shows the graph of more complex objects. The third level is the image itself. Because of our simple Blocks World images and to illustrate algorithm oper-



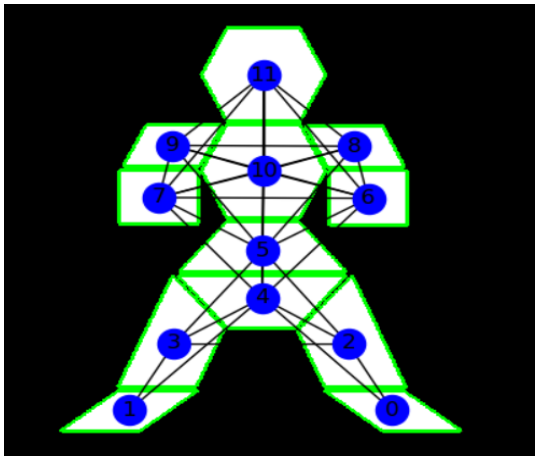
(a)



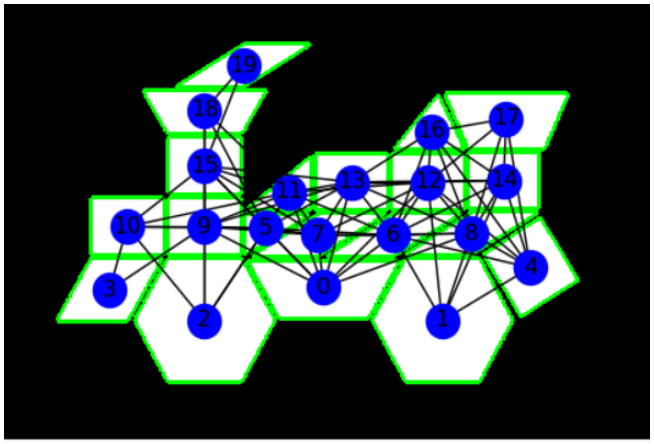
(b)



(c)



(d)



(e)

Figure 3.4: Detected shapes and generated image graphs (level 1) with one connected component.

ation, we assume three levels of hierarchy. However, the number of levels can be increased with the complexity of an image.

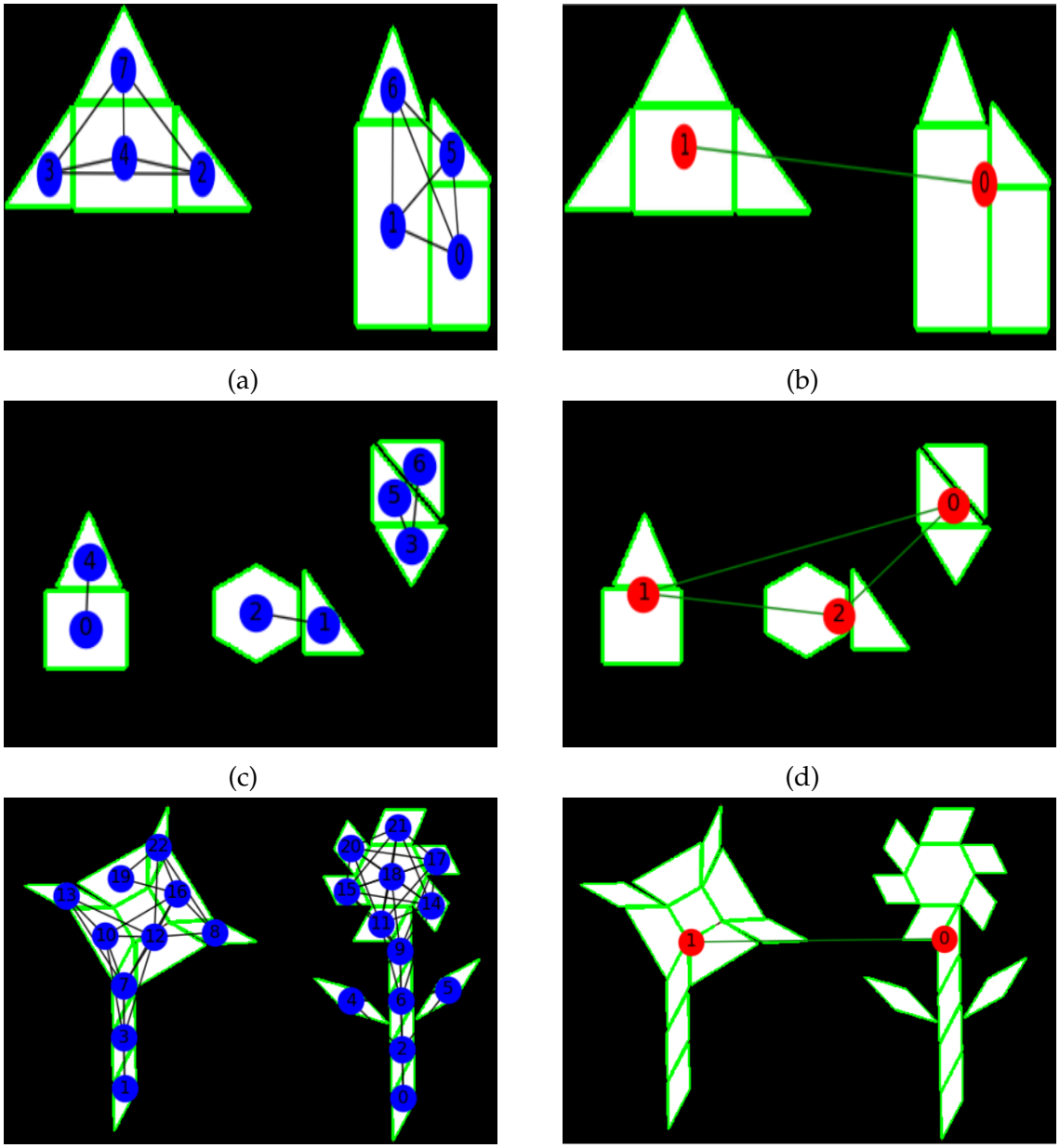


Figure 3.5: Detected shapes and generated image graphs with more than one connected components; (a), (c), (e) Level 1 image graphs with two, three, and two connected components respectively; (b), (d), (f) Level 2 image graphs by considering image components as different objects.

Chapter 4

Sparse Distributed Representation

4.1 Introduction and Motivation

The concept of representing large data in a form of a sparse vector is inspired by the neocortex. The number of neurons in the neocortex is in billions. However, only a low percentage of neurons are highly active at a time. It is believed that brains represent information using a method called Sparse Distributed Representations (SDR) [13]. They represent and store the neurons in a sparse vector, 1 for active neurons, and 0 for inactive neurons. In this chapter, we will encode the image information into SDRs. It is done in a bottom-up hierarchical manner. Attributes of image parts and their neighbors are encoded into SDRs. We perform a Boolean OR to get the object and image SDRs. After that, we can perform graph matching using the image SDRs.

Sparse Distributed Representation is a data representation with special properties for the probability of mismatches, robustness in noise, sub-sampling, classifying vectors, and unions. It is a binary vector where only a few bits are active represented by 1. The active bits can vary from 1% to a few. Each bit generally carries some semantic meaning, so if two SDRs have more than a few overlapping 1's, then those two SDRs have similar meanings [41]. We can encode any type of data into an SDR while observing this aspect of the data. However, there is no sin-

gle fixed approach to encoding ("sparsifying") the data into an SDR. An effective encoder should capture as much information about the data as possible, which will be different for different types of data. Scott [41] discusses several objectives, which should be considered while encoding the data, and it also presents a few encoder examples.

Below are some of the SDR properties which are useful for our work [14]:

1. **Overlap:** To determine the similarity between two SDR vectors, we compute the dot product. The number of bits 'ON' in both the vectors in the same locations is the overlap score.

$$overlap(X, Y) \equiv X.Y \tag{4.1}$$

2. **Matching:** A match between two SDRs is realised if their overlap exceeds some threshold θ. If w_x is the number of bits active in X and w_y is the number of bits active in Y then $w_x \geq \theta$ and $w_y \geq \theta$.

$$match(X, Y) \equiv overlap(X, Y) \geq \theta \tag{4.2}$$

3. **Union:** This is one of the most surprising and fascinating properties of the SDRs. We can store a number of vectors in a single SDR by simply taking a Boolean OR of the vectors. The size of the final vector is the same as the other vectors with a few more active bits. Because of their sparseness there are typically few overlapping bits which would lead to loss of information.

   Example: Let's consider that we have 3 vectors represented as x_1, x_2 and, x_3.

To store these vectors into one X, we take the Boolean OR of all these vectors.

$$x_1 = [0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1]$$

$$x_2 = [1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0]$$

$$x_3 = [0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,1\,0\,0\,0]$$

$$\implies X = x_1\ OR\ x_2\ OR\ x_3$$

$$X = [1\,0\,1\,0\,0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1]$$

This property can also be used in classification where we have c (number of classes) OR'ed SDRs and to determine the class of a new vector, we compute the overlap score and realises a match if it exceeds the threshold.

Example: Let's assume we have 3 classes and there 3 union'ed SDRs X_1, X_2 and, X_3. We have a new test vector y. The number of active bits in y is $w = 2$. For the exact match, we put the threshold $\theta = w$.

$$X_1 = [1\,0\,1\,0\,0\,0\,1\,0\,1\,0\,0\,1\,0\,0\,1]$$

$$X_2 = [0\,1\,1\,0\,0\,1\,0\,0\,1\,0\,1\,0\,1\,0\,0]$$

$$X_3 = [0\,0\,0\,1\,1\,0\,1\,0\,0\,1\,0\,0\,1\,1\,0]$$

$$y = [1\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0]$$

We calculate the overlap scores of y with X_1, X_2 and, X_3. The overlap score

with X_1 is 2 = θ. Therefore, vector y belongs to class 1.

$$overlap(X_1, y) = 2,\ overlap(X_2, y) = 1,\ overlap(X_3, y) = 0$$

By taking union, there is no risk of false negatives and a small probability of false positives. However, the probability of false positives increases while dealing with a sufficiently large number of vectors. If the size of the vector is n, the number of vectors to be stored in the set is M and the number of active bits in the new vector is w then the probability of false positives is,

$$p = (1 - (w/n)^M)^w \tag{4.3}$$

4. **Uniqueness and Exact Matches:** The number of unique SDRs with fixed n and w is,

$$\binom{n}{k} = \frac{n!}{w!(n-w)!} \tag{4.4}$$

If we were to check whether two SDRs with same parameters (n, w) are identical (exact match), the probability of this is,

$$P(x = y) = 1/\binom{n}{k} \tag{4.5}$$

5. **Storing in a memory:** SDR vectors are very large and sparse, most of the bits are 0's. Therefore, to store the whole vector as it is in the memory would be inefficient. To store it in a compact and efficient way, we use sparse matrix techniques and only store the locations/indices of the active bits. **Example:**

$$x = [0\,1\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\ \ldots]$$

$$x' = [2\ 8\ 17\ \ldots]$$

x is the original vector and x' is the vector to store into the memory.

4.2 Related Work

Sparse distributed representation is a way to leverage cortical techniques by which we can process our data in an efficient way. Numenta [13] has been using SDRs for a long time and is continuously improving the procedure and application. SDR has a number of mathematical properties which are aligned to biological intelligence [12]. Further, it has been used in their Hierarchical Temporal Memory (HTM) [13], [42]. HTM is a cortical simulator that processes the input data from multiple streams for various applications such as anomaly detection [11], [43], classification, etc. SDRs are being used in Computer Vision, Neuromorphic Computing and many more areas [44]–[46].

The exploration of SDRs has been a topic of significant interest. Kanerva [47] proposed Sparse Distributed Memory in his 1988 book as a model of human long term memory. He modeled the architecture that could store the large sensory patterns and retrieve them based on partial matches to the input. Denning [48] efficiently summarizes the architecture and properties of Sparse Distributed Memory in the 1989 paper. Later in 1993, Kanerva [49] describes the Sparse Distributed Memory and relates it to other models and circuits. The representation was used for robot control navigation [50] and reinforcement learning [51]. Sensory auditory and visual features are also expressed in sparse representations [52], [53].

4.3 Approach

In the previous sections, we constructed an image hierarchical graph and computed the attributes of the objects in the image. Here, we will discuss the representation of the image objects. In the hierarchy, the SDRs are determined in a bottom-up manner. Sharing representations in a hierarchy leads to a generalization of expected behavior. The corresponding graph hierarchy patterns learned at each level are reused when combined in novel ways at higher levels [13]. The higher levels inherit the properties of lower level components. It makes the computation faster and also reduces memory requirements.

First, we compute the SDRs for the lowest level and then take a union of them to form the higher levels. As we mentioned earlier, we are only considering three levels in this paper. The technique will definitely handle more levels, but as the number of levels increase, the false positives increase. 3 is a reasonable compromise and is more than adequate for Blocks World images. For level one, all the detected contours in the image, which are the components at that level, will have a separate and distinct SDR. The fields and length of the SDR are fixed for all the nodes and levels. We compare and operate on SDRs bit-by-bit, with each bit having a semantic meaning so we do need the SDRs of the same dimensionality.

The significance of SDR in a graph is that a single node's SDR will be able to store its own information as well as its neighbors'. The neighbors are defined from the one-hop connectivity. While designing the encoder, we fix the number of nodes a node can be connected to. In this thesis, we design an encoder, which encodes and stores the graph nodes' attributes into the SDR. Here we will be dealing with the block polygons in a simple 2D "blocks world" image space.

The two considerations for encoding the data into the SDR are described below:

1. SDRs should be sparse. The sparsity for encoders can vary but should be relatively fixed for a given application of an encoder [41]. A very rough rule of thumb is that the number of 1's should be the log_2 of the dimension. For this, we assign each field of an SDR a fixed number of bits assuming b and keeping only w bits ON. This way, each dimension is sparsified by a w/b factor.

2. The use of SDRs should be mostly independent of the indexing scheme representing the graph, for example, the adjacency list or matrix. A single SDR should have a reasonable knowledge of its surroundings, regardless of the predefined indexing. Having a certain level of independence in the indexing is important for the usefulness of the SDR for pattern matching. When storing the neighbors' attributes into the SDR, we process them in the clockwise direction, keeping a particular, invariant, geometric coordinate as the reference.

This way, we can compare the SDRs of two different nodes and find similarity metrics between the two. Therefore, as desired, the usefulness of SDRs becomes independent of how the nodes in the graph were originally indexed. However, this approach does limit rotation invariance. Though few systems (including biological vision) provide robust rotation invariance.

The attributes are defined as the number of edges, the height-width ratio, and connectivity (number of neighbors). To store the relative positions of neighbors, we compute an angle between the node and the neighbor node. An SDR has two information fields, one for the node and other for the neighbors as described in Figure 4.1. The final SDR of a node has $m + 1$ fields, one for the node and m for the neighbors. Each field has sub-fields to store that node's attributes and the neighbor

node's relative positions.

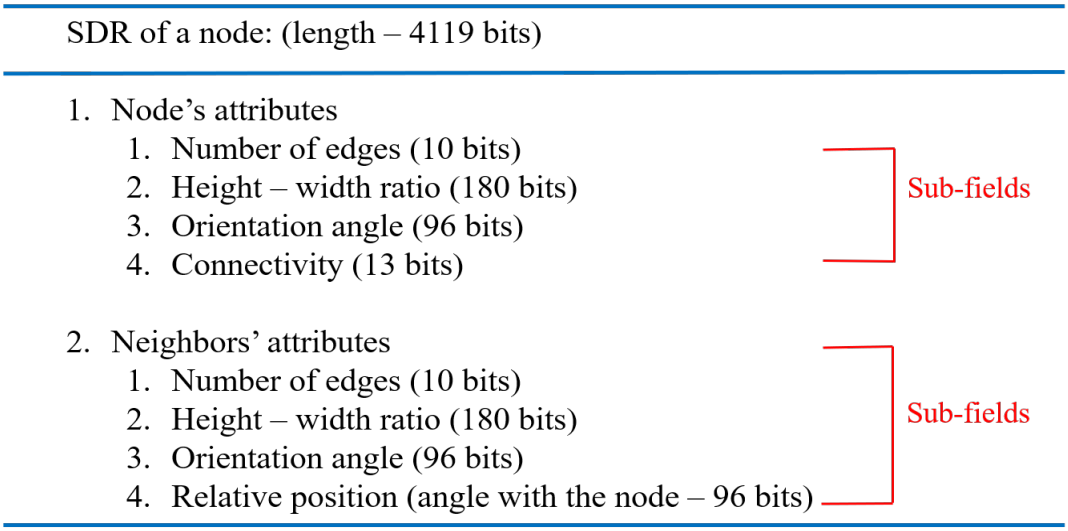


Figure 4.1: A graph node's SDR organization [1].

To understand the encoder algorithm, assume a 2D planar labeled graph, G, having n nodes, where every node is connected sparsely to, at most m, other nodes, and the nodes are labeled with their attributes (such as number of edges, height - width ratio, orientation and connectivity, information that is easily attainable from OpenCV). Each node will have a distinct SDR. The length of the each SDR field and the number of active bits are fixed.

To encode individual attribute information in SDR fields, we use flocet codes. We can understand the flocet code from the sliding window in the fields. Let's assume that we have a sliding window of a specific size. Instead of just activating one bit, we would set some consecutive bits active from an index. And, this whole window moves by one place to the next bit. We split the range of numerical values into buckets and then map the buckets to the values. From the bucket index itself, we set w consecutive bits 'ON'. The algorithm for determining the bucket a value falls into is presented in algorithm 4.1 [41].

Algorithm 4.1 Value Encoder

Input A numeric value v (It can either be height-width ratio, orientation angle or relative position), minimum range $minV$, maximum range $maxV$ and, the number of buckets nb

Output Bucket index i, the value falls into.

procedure ENCODER($v, minV, maxV, nb$)

/* Calculate the range of values */
$rangeV = maxV - minV$

/* Compute bucket index i */
$i = floor(nb \times \frac{v - minV}{rangeV})$

return i

As we see in figure 4.1, there are 5 different types of attributes in an SDR, (1) Number of edges, (2) Height - width ratio, (3) Orientation angle, (4) Connectivity and, (5) Relative position. (1) and (4) are smaller in size. We choose the flocet size 3 for these two fields and 6 for the three other fields as the number of active bits in them. Therefore, the criteria for encoding these into SDR fields are different.

1. For the number of edges and connectivity of the nodes, we assume s and c bits respectively. This means that we are limiting the maximum number of sides a polygon can have and to how many other nodes it can be connected. Whatever the values of number of edges and connectivity are, we set 3 (sliding window size) consecutive bits 'ON' from that value among the total s and c bits. The sparsity of these fields is $3/s$ and $3/c$ respectively.

2. For height-width ratio and angle fields, we assume $2b$ and b bits respectively. Starting from the calculated index i from algorithm 4.1, we will take 6 (sliding window size) consecutive bits and set them 'ON'. This makes the sparsity $6/2b$ and, $6/b$. For the height-width ratio and angle, we set the range from 0 to 360. Here, we choose the number of buckets nb to 4, and the field length

becomes 90. There is a trade-off on bucket size. The larger the bucket, the higher the probability of false positives, less specific values, but the more efficient the encoding.

Example: Let's assume a graph where one of the nodes is connected to 2 other nodes. The node is a triangle object and connected nodes are rectangle and pentagon objects. Figure 4.2 represents the SDR of the node (with individual fields shown here). We assume that a triangle is connected to a rectangle in an image. The vector shows the information of the triangle node and its neighbor rectangle node. In the representation, we assume a node can have a maximum of 10 neighbors. If a node has less than 10 neighbors then all the other neighbor fields are all 0's. These fields are not shown in figure 4.2.

Node				Neighbor			
Number of edges	Height – width ratio	Orientation angle	Connectivity	Number of edges	Height – width ratio	Orientation angle	Relative position
00111...00	00...111111...00	0111111...00	01110...00	000111...00	0111111...00	00...111111...00	00...1111110

Figure 4.2: A node's SDR. Two fields for the node and its neighbor with flocet codes in the sub-fields. Trailing 0's are not shown here.

Adding more bits per field yields more accuracy, fewer bits lumps more figures into each field. Different applications may have different preferences. We choose 10 as the maximum number of edges. This could also be more than 10. Humans are not going to be able to easily tell an object with 10 neighbors from an object with 11 neighbors, probably not 9 and 10 either. You can say the same thing about number of edges and orientation angle, at some point a human can't tell the difference. So the numbers of bits (resolution) and maximum values can be related to the limitations of human vision. One could also think of a logarithmic encoding that emphasises smaller number of edges vs. large number of edges.

Sparse distributed representation for higher levels

After calculating the SDRs for each and every node of a level 1 graph, we move up to the hierarchy. For level 2, to determine the SDRs of the nodes, we combine the SDRs of level 1 by taking the connected component nodes. We perform 'union' operations for every node present in level 2 and these union SDRs represent a hierarchical graph's structure. For example, assume we have a graph that consists of nine objects and three connected components. We compute three SDRs for level 2 by performing 'union' operations on the objects which belong to the connected components. These three SDRs are the fixed-length binary representation of a level two graph.

By the union property, a single SDR is able to store a dynamic set of elements, so when we see the final SDR after performing the union, it has the information presented in the component node SDRs. We can also represent the whole graph in a single SDR by taking the union of all its nodes' SDRs. This resultant SDR will have relevant information about the graph and represents our level 3, which is the entire image. Even if we have more than three levels in the graph we still only require this bottom-up approach: calculate the SDR for the lowest level and then start combining (union operation) the SDRs for higher levels motivated by the approach our brain takes when processing a piece of new visual information.

The SDRs of image-graphs have four important characteristics, which allow them to achieve their goal of fast pattern matching in graphs.

1. Each bit in an SDR has semantic meaning.

2. Computations with SDRs are independent of the indexing in graphs and their components.

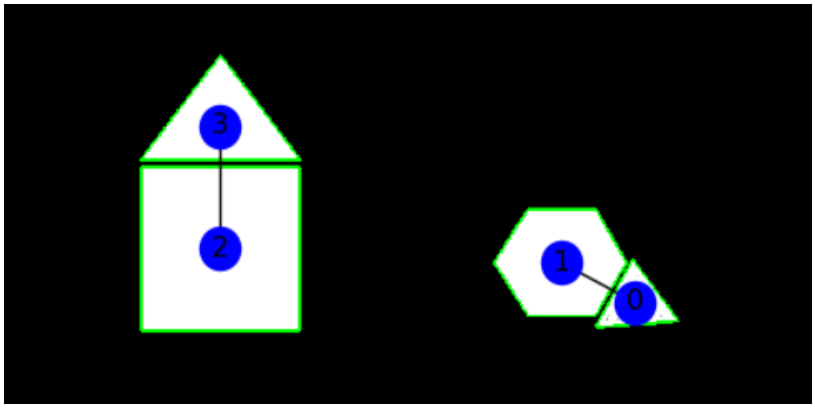
3. SDRs are sparse enough to reduce spatial complexity.

4. The SDRs form a representation that contains the "structure" of the object and so is useful in downstream object recognition.

4.4 Results

The generated SDRs are large binary vectors representing the important attributes of the objects. Each detected part in the image has an SDR of length l. The length of the SDRs is large compared to the number of active bits. For limiting the size of the SDR, we assume that the maximum connected nodes and the maximum number of edges for a node are 10. The height and width can be in a range from 1 to 360. To represent the very sparse SDRs, we show only the indices of ON bits. The computation with SDRs is memory and time efficient as the computation happens only with the active bits.

Figure 4.3 shows a simple blocks-world image, and it's corresponding graph and SDRs. Blue nodes are the centers of detected parts from OpenCV. Here, we show individual part nodes' SDRs to illustrate the encoding of information. Initial bits from 0 to 299 represent the node information and from 300 to 4119 bits represent the connected neighbor nodes' information and their relative position with the node.



(a) A simple level 1 image graph with four parts. (0, 1) and (2, 3) are the connected components and considered different objects for level 2 graph.

Part 0 SDR:
[3 4 5 100 101 102 103 104 105 204 205 206 207 208 209 287 288 289 305 306 307 399 400 401 402 403 404 489 490 491 492 493 494 636 637 638 639 640 641]

Part 1 SDR:
[6 7 8 100 101 102 103 104 105 190 191 192 193 194 195 287 288 289 302 303 304 399 400 401 402 403 404 503 504 505 506 507 508 591 592 593 594 595 596]

Part 2 SDR:
[4 5 6 100 101 102 103 104 105 190 191 192 193 194 195 287 288 289 302 303 304 399 400 401 402 403 404 489 490 491 492 493 494 585 586 587 588 589 590]

Part 3 SDR:
[3 4 5 100 101 102 103 104 105 190 191 192 193 194 195 287 288 289 303 304 305 399 400 401 402 403 404 489 490 491 492 493 494 585 586 587 588 589 590]

(b) Level 1 SDRs for the four parts of image. Only the indices of active bits are shown here.

Figure 4.3: Level 1 image graph and SDRs of the parts (blue nodes) are shown here. Each part has only one neighbor node. Initial bits from 0 to 299 represent the part information and the rest bits describe the neighbor information.

Chapter 5

Graph Matching

5.1 Introduction and Motivation

In this thesis, our goal is to recognize the pattern in an image by constructing the image graph and then match the patterns of images by doing the graph matching. Graph matching is done by comparing their SDRs. Graphs are used to encode the structural information of images. Graph matching algorithms are key in the pattern recognition field [54], [55].

Graph matching is a problem of finding the similarity between two graphs. There are two types of matching, exact matching or inexact matching. Exact graph matching is also known as the graph isomorphism problem. The sub-graph isomorphism problem is also called exact graph matching. In graph isomorphism, we determine whether two graphs are isomorphic and in sub-graph isomorphism, we determine whether the smaller graph is graph isomorphic to a part of the bigger graph. Two graphs G_1 and G_2 are said to be isomorphic if (a) their number of components (vertices and edges) are same, and (b) their edge connectivity is retained. There exists a bijection function f from vertices of G_1 to vertices of G_2, $[f : V(G_1) \implies V(G_2)]$ then $G_1 \simeq G_2$. Subgraph isomorphism is an NP-complete problem. Figure 5.1 shows three graphs which are isomorphic to each other. Two graphs G_1 and G_2 are said to be subgraph isomorphic if, for a subgraph G_0 of G_1,

$G_0 \subseteq G_1$, there exists a bijection function 'f' from vertices of G_0 to vertices of G_2, $[f : V(G_0) \implies V(G_2)]$. Figure 5.2 displays two graphs and the node to node matching between them. It shows that the smaller graph is present in the bigger graph.

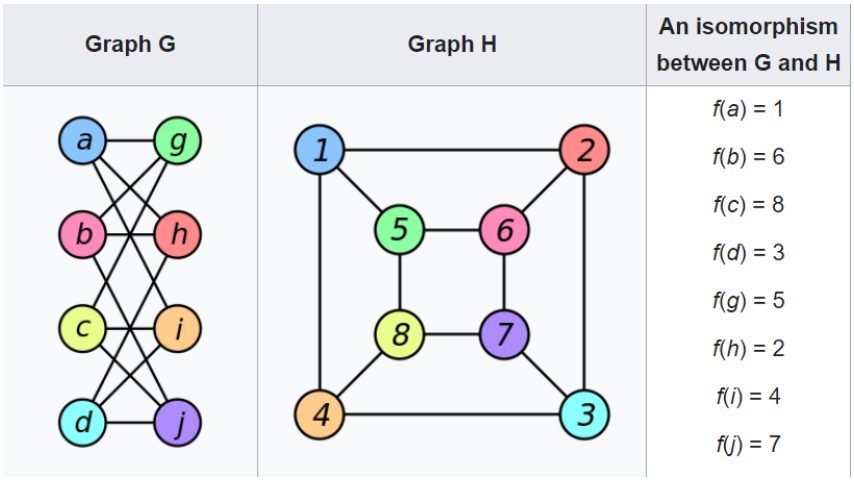


Figure 5.1: Two isomorphic graphs are shown here with their bijection function mapping f [56].

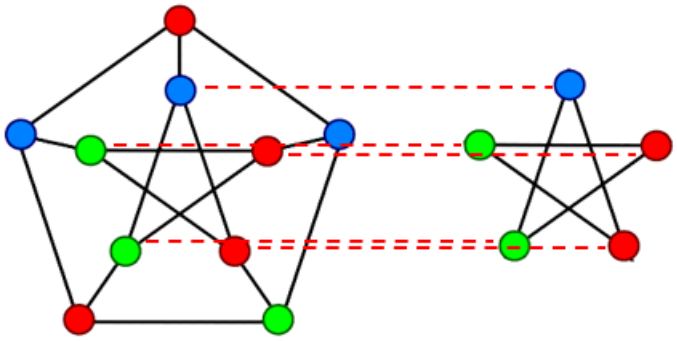


Figure 5.2: A subgraph matching; the object graph is matched to the scene graph.

5.2 Related work

Object recognition is the primary operation of any computer vision system. One obvious method of recognizing an object is by comparing it to a database of known objects, template matching is an example of this approach. One way to incorporate more flexibility into the recognition process is to represent objects by graphs, which incorporate the structural information in the image. For example, in computer vision, graphs have been shown to be a useful tool for representing images. Labeled graphs can capture and represent a significant amount of information on the "structure" of objects. Using graphs, object recognition requires graph matching [34], [37], [38], [57], [58].

5.3 Approach

Graph isomorphism, which is also known as exact graph matching is used in the area of image recognition. This problem is known to be solved in non-polynomial time, but here we are proposing a new method for solving approximate graph isomorphism to reduce the complexity of pattern matching by combining graph analytics and sparse distributed representations. The algorithm is heuristic. Graph isomorphism can only be applied when the number of nodes in the graphs are the same. Also, two graphs cannot be isomorphic if the distribution of in-degree/out-degree of their nodes are different, e.g., 3 nodes with 2 connections, 2 nodes with 4 connections. Therefore, we check the number of nodes in the graphs' level representation, if equal; we check the isomorphism between the level 1 SDRs. If not isomorphic, we move to level 2 and calculate the sub-graphs of the bigger graph. The sub-graphs respect the hierarchy. We check the isomorphism for all the sub-graphs whose number of nodes are equal to the smaller graph's nodes. If the smaller graph

exists in the bigger graph, the graph is sub-graph isomorphic.

The computational savings come at the cost of capturing and representing more complete information in the SDR. Although, SDR vectors are large, the operations using SDRs depend on the number of active bits, which are much fewer than the total number of bits. This is an advantage of sparse representations. SDR vectors contain most of the information about the objects' geometries and the structure of an image. More information can be added based on the application and the dataset. Adding information improves robustness. However, this comes with the cost of more false positives. False positives are possible but for very sparse encodings very unlikely [14]. We realize the match between the SDRs using SDR's union property and a threshold θ. Decreasing θ also results in more false positives. One advantage of the union property is that there is no risk of false negatives since the overlap gives the perfect match if the SDR is within the set. However, it does increase the chance of false positives [14], by increasing the number of active bits in the resultant SDR.

With the help of SDRs, we have developed a powerful heuristic search for graph isomorphism in $O(l)$ time, l is the SDR length which is a constant in our case. A variation of exact match isomorphism is called subgraph isomorphism. Here one must determine whether a graph contains a subgraph, which is isomorphic to another graph. This problem is also known to not be solvable in polynomial time. Here, we choose k nodes' subgraph out of a big graph of n nodes in $O(n^k)$ time and with the help of SDRs, do the matching in $O(1)$. All the k nodes' subgraphs should respect the hierarchy (shown in figure 3.3). k is the number of nodes in the small graph. For efficient image matching, an SDR should be invariant to position, scale, brightness and, rotation of an object. In this paper, our SDR provides both scale and position invariance. The graph-matching algorithm using

our SDR is shown in algorithm below. In the future, we can apply this technique of merging graph matching and SDRs to find a solution for probabilistic matching, by, for example, finding matching patterns in an image using probabilistic associative memory, which is known to approximate Bayesian Inference.

Algorithm 5.1 Graph Matching

Input First level image graphs (G_A, G_B) and their 2D SDR arrays (S_A, S_B) with n_A and n_B nodes.

Output Whether graphs are matched or not. If matched, they are isomorphic or sub-graph isomorphic.

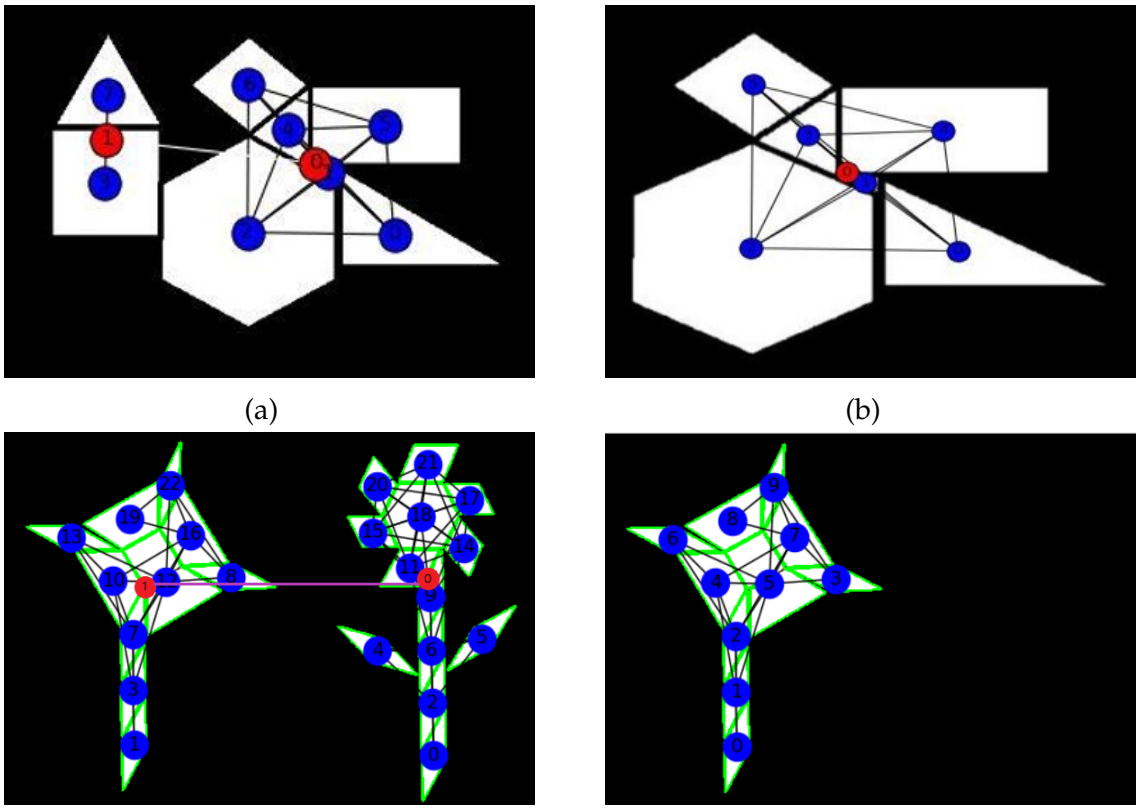
```
1:  procedure GRAPHMATCHING
2:
3:      /* If number of nodes are equal - check graph isomorphism*/
4:      if n_A == n_B then
5:          Ans ← GRAPHISOMORPHISM(S_A, S_B)
6:
7:      /* else - check sub-graph isomorphism */
8:
9:      /* If G_A has more nodes then G_B */
10:     else if n_A > n_B then
11:         calculate level 2 graph G_A2 and a 2D SDR array S_A2 for graph G_A
12:
13:        /* Iterate through all the sub-graphs of G_A. */
14:        for i in len(S_A2) do
15:
16:          /* If the number of nodes are equal to of G_B then check for
17:           isomorphism */
18:            if number of nodes in S_A2[i] == n_B then
19:                Ans ← GRAPHISOMORPHISM(S_A2[i], S_B)
20:            end if
21:        end for
22:
23:     /* If G_B has more nodes then G_A */
24:     else then
25:         calculate level 2 graph G_B2 and a 2D SDR array S_B2 for graph G_B
26:
27:        /* Iterate through all the sub-graphs of G_B. If the number of
28:           nodes are equal to of G_A then check for isomorphism */
29:        for i in len(S_B2) do
30:            if number of nodes in S_B2[i] == n_A then
31:                Ans ← GRAPHISOMORPHISM(S_A, S_B2[i])
32:            end if
33:        end for
34:     end if
35:
36:     if Ans == 'Yes' then
37:         "Graphs are sub-graph isomorphic"
38:     else then
39:         "Graphs are not sub-graph isomorphic"
40:     end if
41: end procedure=0
```

```
1: procedure GRAPHISOMORPHISM
2:     calculate union of the SDR arrays and create 1D SDRs
3:     Take dot product of S_A and S_B to get the overlap score for the SDRs to
4:     determine the similarity.
5:     We realise a match between the SDRs if their overlap exceeds some
6:     threshold θ.
7: end procedure
```

5.4 Results

In this section, we calculate the match between the generated graphs using SDR overlap. In figure 5.3, we take two sets of graphs and check whether the first graph contains a graph which is isomorphic to the second graph. This demonstrates whether the object present in the second image exists somewhere in the first image. Here we also show that this check is independent of the graph/object indices. As one can see in the images, some of the detected parts in the second image are indexed differently from the first image, which does not affect the final result. This match also demonstrates the scale and position invariance. For the graphs in figure 5.3, image (a, c) and image (b, d) have two and one objects respectively in level 2 which are represented by a red node. We take one object SDR of graph 1 at a time and compare it with the graph 2 SDR, which realizes a match. For the given images in figure 5.3 the SDR overlap exceeds the threshold. We conclude that graph 2 is sub-graph isomorphic to graph 1 which also means that the object in the second image exists in the first image. It should be noted, that such matching can be done in a straightforward manner by cortical-like associative memories as we will show with Sparsey in chapter 6.

Here, we explain figures 5.3 (c) and (d) to show their SDRs and matching. In figure 5.3(c), components {0, 2, 4, 5, 6, 9, 11, 14, 15, 17, 18, 20, 21} of level 1 belong



(a) (b)

(c) (d)

Figure 5.3: Graphs with sub-graph isomorphism, right images (b) and (d) are smaller graphs which are isomorphic to a part of the left bigger graphs (a), (c) respectively.

to object 0 in level 2 and components {1, 3, 7, 8, 10, 12, 13, 16, 19, 22} of level 1 belong to object 1 in level 2 graph. Using algorithm 1, we first check the number of nodes in both the image graphs. Figure 5.3(c) has two nodes and 5.3(d) has one object. Therefore, we take all the sub-graphs of 5.3(c). There is only one sub-graph which has equal number of nodes as 5.3(d). Hence, we match the SDR of object 1 of 5.3(c) and SDR of 5.3(d). The match exceeds the threshold of 90%. The threshold is determined by the number of active bits in both the graphs. Thus these two graphs are sub-graph isomorphic.

Table 5.1: Result Analysis: Techniques and their complexity

Algorithm	Complexity
Graph Isomorphism	NP-Intermediate
Sub-graph Isomorphism	NP-Complete
Approximate Graph Isomorphism w/SDR	$O(1)$
Approximate Sub-graph Isomorphism w/ SDR	Choosing a k node subgraph out of a big graph with n nodes – $O(n^k)$ and matching subgraph with k nodes is $O(1)$

Chapter 6

Retrieval of Information From Noisy or Incomplete Data

6.1 Introduction and Motivation

In this chapter, we will use the SDRs to recognize blocks-world images. We retrieve the images from a noisy version of it. Most humans can recall information from an incomplete or a noisy variation of it. Let's assume we remember the sentence "All work and no play makes Jack a dull boy" then we can recall it from "All work and no play makes" (incomplete) and "All work and no play makes Jack a happy boy" (noisy).

There are two kinds of noise we want to investigate. There are the usual sources of noise, such as speckle and partial occlusion, and there is Shape Noise, which is a different kind of noise: resulting from missing or spurious and poorly placed components. Using SDRs to retrieve shape information helps us in assessing how much information SDRs capture of objects and images. The approach is unique, and there isn't any competing data that we can compare.

6.1.1 Associative Memory

Associative memory is a Content-Addressable Memory (CAM) that allows the retrieval of data based on the similarity of input data with stored data in the memory. There are two general classes of Associative Memory: exact match and best match.

Exact match is used extensively in computer science (caches do exact match on memory addresses), database systems, IP address lookup, etc. It is straight forward to do, say with hashing. The best match finds the closest match if an exact match is not possible. Best match association can be done in a probabilistic manner. It is more complex and expensive and used less often. Here, we look at best match association of which there is both auto-associative memory and hetero-associative memory.

Auto-associative Memory: In auto associative memory, we retrieve the content/pattern already stored in the memory from the noisy/incomplete input data. The pattern in the memory which is closest to the input pattern would be the output. Figure 6.1 shows an auto-associative model. We can see that there are 4 shapes stored and when we input a noisy shape, it returns the original corresponding shape without noise. Most humans can recall the missing information from a portion of data.

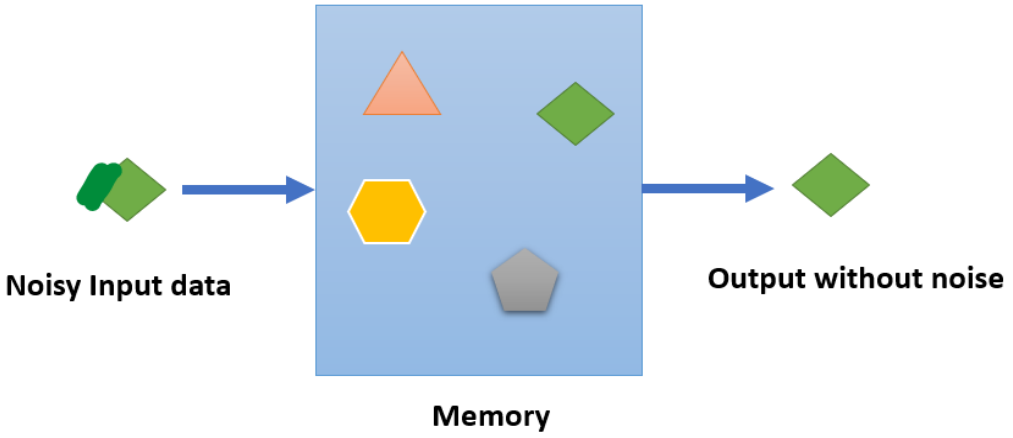


Figure 6.1: Auto-associative memory model as a black box. Four shapes are stored in the memory. When a shape is input, it retrieves the closest matched shape from the memory.

Figure 6.3 shows an example of how the auto-associative memory stores the

original data and recalls it from noisy/incomplete data. Here, we have $x = y$ as there is no associated data/pattern. Therefore, the weight matrix is calculated as equation 6.1. X is orthonormal matrix of all stored bit vectors and p is the number of bit vectors stored the memory. To retrieve the vector o from the memory using noisy/incomplete vector t the operation can be written as equation 6.2 [59].

$$W = XX^T = \sum_{i=0}^{p} x_i \times x_i \tag{6.1}$$

$$o_j = \sum_{i} W_{ij} t_i \tag{6.2}$$

Hetero-associative Memory: In hetero-associative memory, we retrieve the closest associated content/pattern to the input data. The output is different from input not only in content, but also in type and format. For auto-association, the input and output spaces are the same (equation 6.1) whereas for hetero-association, $X \neq Y$ which could be of different dimensions. Therefore, the weight matrix is calculated as equation 6.3. Using the generated weight matrix, we retrieve the associated data from the memory using equation 6.2.

$$W = XY^T = \sum_{i=0}^{p} x_i \times y_i \tag{6.3}$$

Figure 6.2 shows a hetero-associative model. There are 3 sets of shapes, when we input a shape, it returns the associated shape as output.

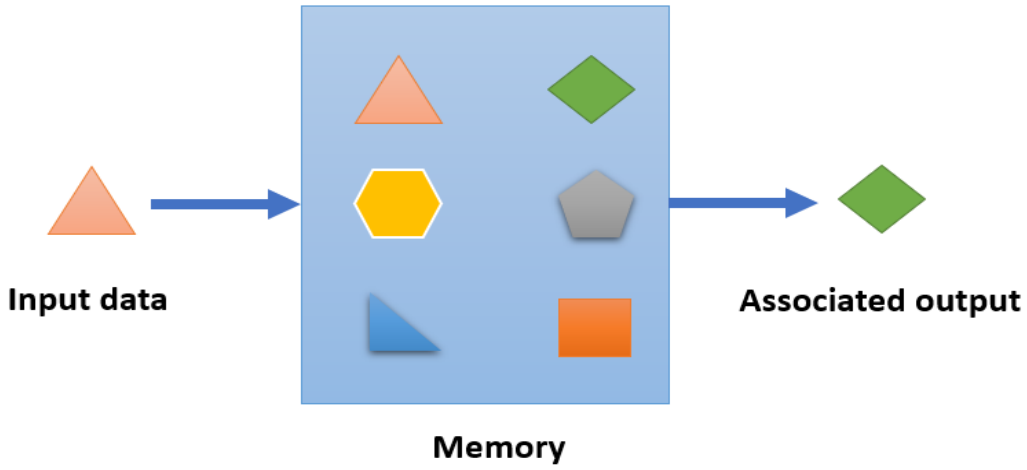
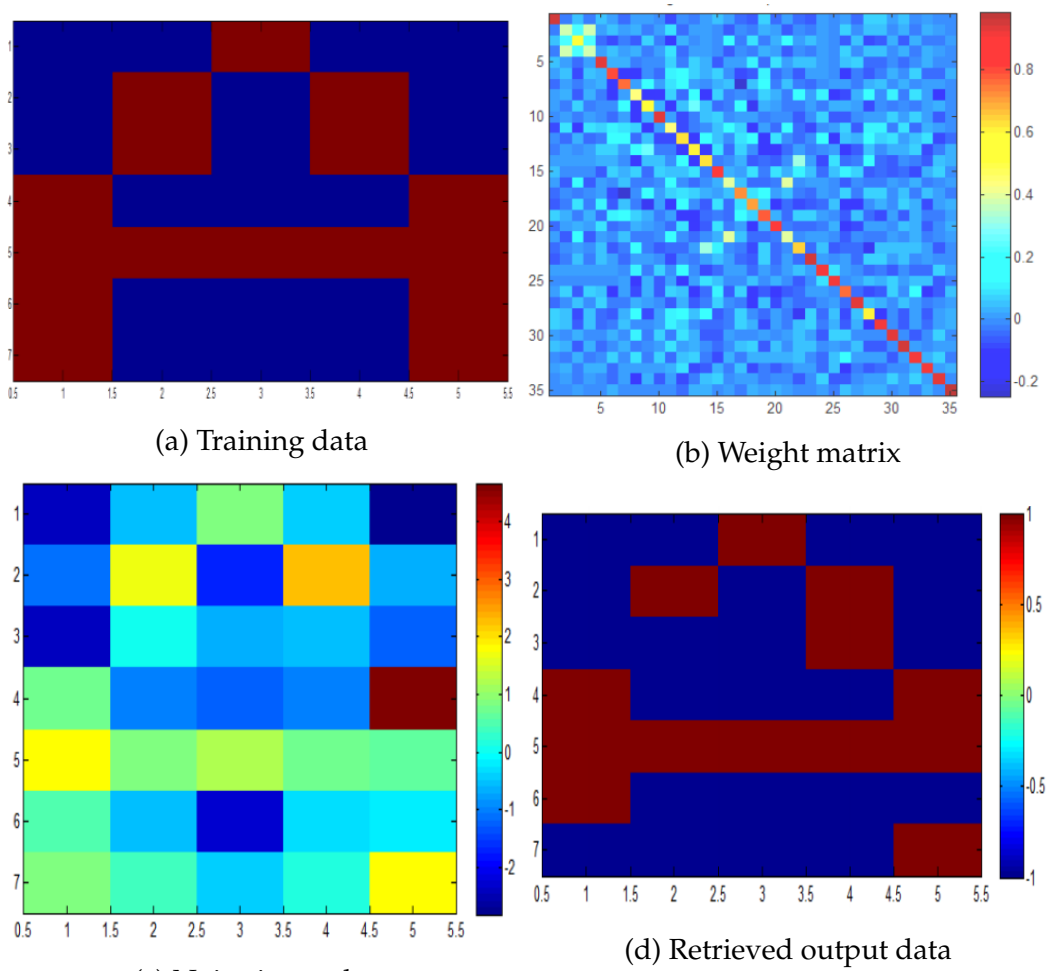


Figure 6.2: Hetero-associative memory model as a black box. Three associated pairs of shapes are stored in the memory. When input a triangle, it retrieved the associated diamond from the memory.

6.1.2 Support Vector Machine

The Support Vector Machine (SVM) is a supervised learning model used for classification, regression, and outlier detection. It's objective is to construct a hyperplane or set of hyperplanes in a high dimensional space to distinctly classify the data points [61]. The high dimensional spaces can also be non-linear wrt the original parameters. A hyper plane in a high dimensional, non-linear space can be a complex partition of the original data space. SVMs do not do higher level hierarchical abstraction, they just create complex non-linear partitions. This is ideal for us, since we are trying to determine how much useful information our SDR encoding captures of the original object.

SVM uses a subset of training points in the decision function (called support vectors), so it is also memory efficient [62]. A good hyperplane is found when it has the maximum margin, i.e., maximum distance to the support vector points. Maximizing the margin provides the low generalization error of the classifier [61].



(a) Training data

(b) Weight matrix

(c) Noisy input data

(d) Retrieved output data

Figure 6.3: Recalling of data from auto-associative memory [60].

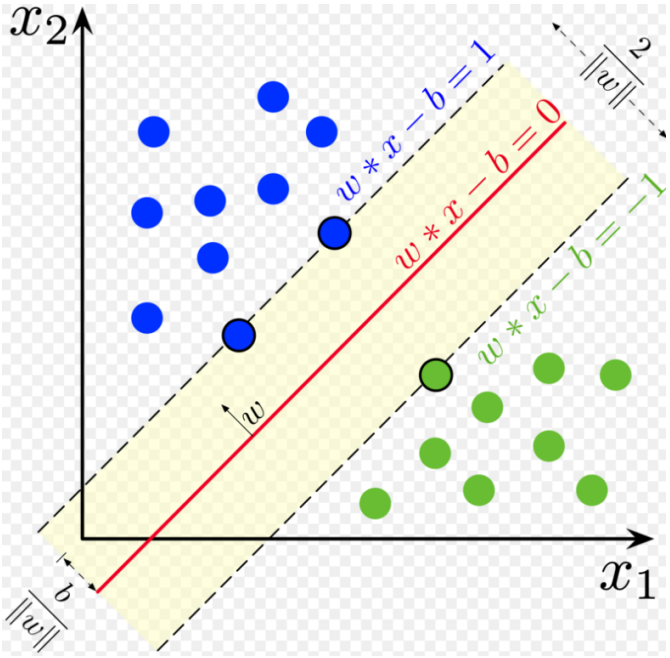


Figure 6.4: Maximum margin hyperplane for two linearly separable classes [61].

For two linearly separable classes, given a training dataset of n points, $(\vec{x_1}, y_1), (\vec{x_2}, y_2), \ldots, (\vec{x_n}, y_n)$, where $y_i = \pm 1$, an hyperplane can be written as equation 6.4 [61].

$$\vec{w}.\vec{x} - b = 0 \tag{6.4}$$

$\vec{w}$ is the normal vector to the hyperplane. An example of linearly classifying data points using SVM is shown in figure 6.4. The parameter $\frac{b}{\|\vec{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\vec{w}$. Two parallel hyperplanes make the region called margin between the two classes. These hyperplanes can be written as equations 6.5 and 6.6.

$$\vec{w}.\vec{x}_i - b \geq 1, \quad \text{if } y_i = 1 \quad \text{and} \quad \vec{w}.\vec{x}_i - b \leq -1, \quad \text{if } y_i = -1 \tag{6.5}$$

$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 \quad \text{for all } 1 \leq i \leq n \tag{6.6}$$

SVM can also efficiently do non-linear classification using the kernel trick as shown in figure 6.5. This operates on the data points in a high-dimensional space. SVM works only when the data is labeled since it is a supervised learning model. SVM works for multi-class classification too. This is done by reducing one multi-class problem into multiple binary problems [61]. SVM uses two approaches for multi-class classification (a) one-vs-one (b) one-vs-rest. In one-vs-one approach, it takes any two classes and form the hyperplane. In one-vs-rest approach, it takes one class as class 1 and all the other classes as class 2. This proceeds iteratively for all the classes present in the input data. In figure 6.6, there are three class input points and both the approaches are explained.

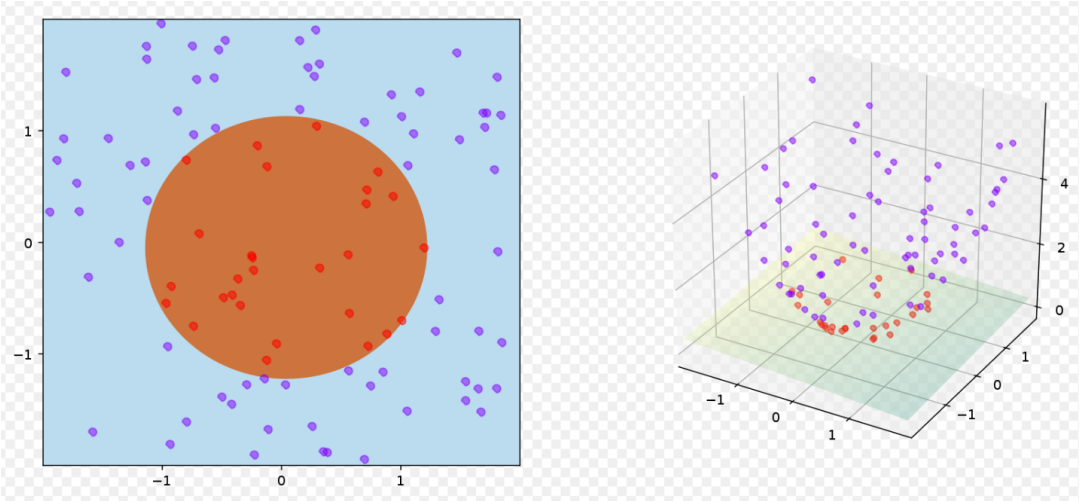


Figure 6.5: SVM with kernel given by $\phi((a,b)) = (a,b,a^2+b^2)$[61]

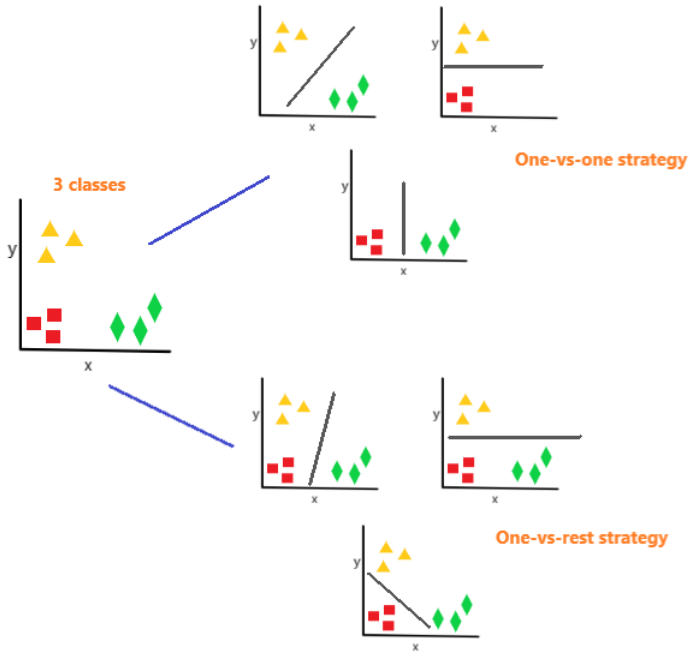


Figure 6.6: Example of multi-class classification approaches to three class points.

6.1.3 Sparsey

Sparsey [44], [45] is a neuromorphic associative memory model where information is represented using a SDR format. It is a hierarchical model with each level consisting of an array of macs (macro-columns). Each mac has three types of connections, bottom-up (U), top-down (D) and horizontal (H). It learns and retrieves the best-match stored sequence in fixed time [63]. In sparse distributed coding (SDC), item (part of an input vector) is coded by a small subset of the mac's units. A single coding field in Sparsey consists of Q Winner-Take-All (WTA) Competitive Modules (CMs). Figure 6.7 shows the connection between input-mac and mac-mac.

A code selection algorithm [45] determines which cells are chosen to represent an input, during both learning and retrieval. While learning the data, it ensures

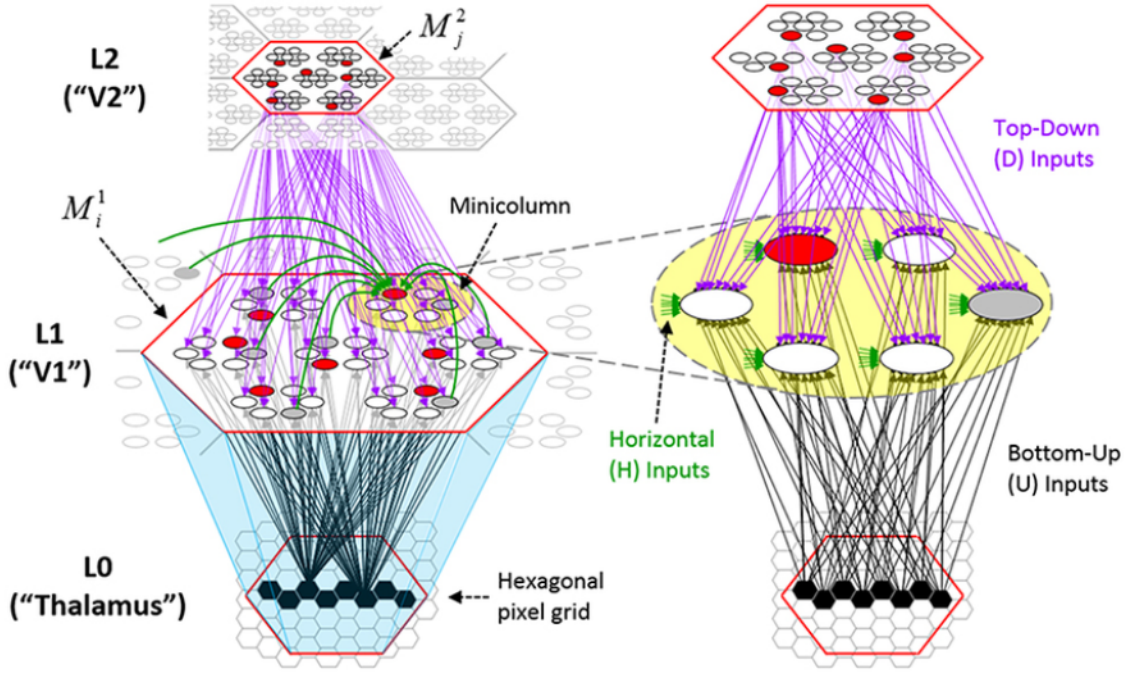


Figure 6.7: Afferent projections to a mac [44].

that similar codes are assigned to similar inputs (SISC property) that leverages similar shape characteristics of our SDR code. When we input a new test pattern, it would assign similar codes if the model has already learned a pattern close to it. If the test pattern is completely unfamiliar to the model, it will assign a new code. Figure 6.8 shows the architecture of input encoding using CLA.

Our hypothesis is that this will open up the space making it easier to distinguish neighboring representations. This hypothesis depends on our assumption that the SDR provides enough information to distinguish different shapes (i.e., resolve shape noise). Sparsey is a re-encoding of the vector space, but it does not add any new information.

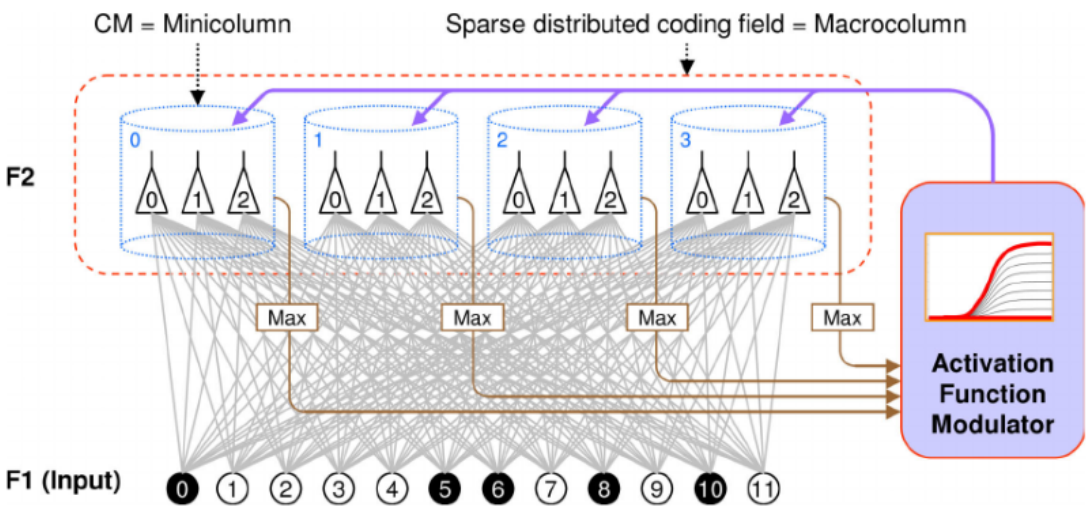


Figure 6.8: Functional architecture of CLA [45].

6.2 Related work

The concept of associative memory has been studied for some time. Around 1960, Steinbuch [64] invented an associative memory like architecture of an artificial neural network. In 1980, Palm [65] presented a model where the information stored in the network is obtained from the retrieved result. He calculates the information storing capacity of associative memories.

Kosko [66] introduced Bidirectional Associative Memory (BAM) in 1988, a hetero-associative memory. BAM transforms the binary data into bipolar form. It could recall a pattern from another pattern which can be of a different sizes. Sparse Distributed Memory (SDM) [47] is another associative memory like model. SDM is sensitive to the similarity of data. It retrieves the word stored at an address with an address close to it. It checks for the similarity of addresses by calculating the hamming distance between them. Zhu [67] investigates several models based on associative memory and proposes a solution of building a hierarchical network of Bayesian Memories to solve the scaling issue of the large networks. Taha [68]

designed a Hamming Distance Associative Content Addressable Memory (HDA-CAM) which leverages in-memory parallel computing and achieves low-power and faster computation.

6.3 SVM Approach

To retrieve the shape information from noisy or incomplete data, we use an SVM. SVM chosen since it needs only a modest number of training vectors to create en effective partition, as opposed to an MLP, which typically requires a fairly large training set, which is difficult for us to us to generate artificially. The training input is the shapes' sparse distributed representations as calculated in chapter 4. Here, an SVM is used for doing multi-class classification. We use 2D blocks world images which have one object made of multiple components (1 connected component in level 1 graph) and consider each image as different class. We calculate image SDRs and train the SVM on these vectors. SVM partitions underlying SDR vector space (i.e. input image SDR vectors) into the classes. This helps us in learning the shapes and structural information of images. Figure 6.9 is an example of a vector space partitioned into 10 classes.

To implement the training of images, we use the scikit-learn machine learning library for Python [62]. The support vector machines in scikit-learn has 3 versions SVC, NuSVC and LinearSVC capable of performing binary and multi-class classification. Both SVC (C-Support Vector Classification) and NuSVC (Nu-Support Vector Classification) are similar methods except for the parameters to control the number of support vectors. For multi-class classification, they always use 'one-vs-one' strategy ('ovo').

The primal problem for the SVC is, given training vectors $x_i \in \mathbb{R}^p, i =$

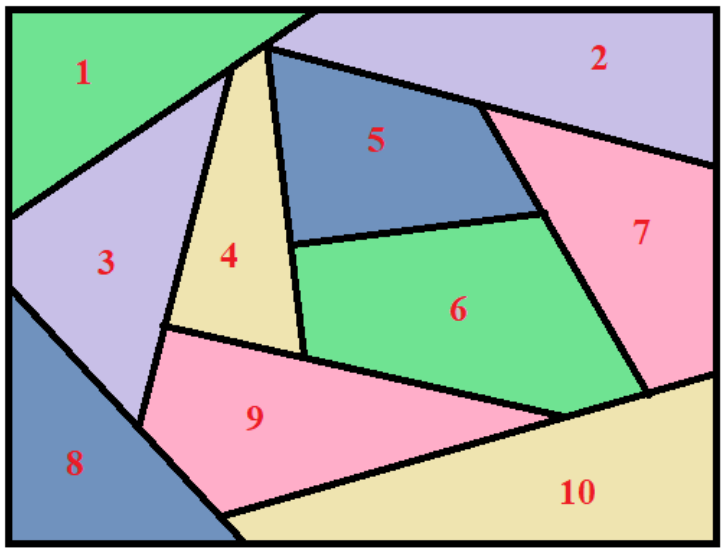


Figure 6.9: Linear partitioning of a vector space into 10 classes

$1, 2, \ldots, n$ in two classes, and a vector $y = \{-1, 1\}^n$, can be written as equation 6.7. The goal is to find $w \in \mathbb{R}^p$ and $b \in \mathbb{R}$ such that the prediction given by $\text{sign}(w^T \phi(x) + b)$ is correct for most samples [69]. ϕ and C are the identity function and regularization parameter respectively. ζ is the distance allowed from samples to their correct margin boundary.

$$\begin{aligned} & min_{w,b,\,\zeta_i} \; \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to } & \; y_i (w^T \phi(x_i) + b) \geq 1 - \zeta_i, \\ & \; \zeta_i \geq 0, i = 1, \ldots, n \end{aligned} \tag{6.7}$$

LinearSVC is faster than SVC and NuSVC. The kernel in this case is assumed to be linear. It implements a 'one-vs-the-rest' ('ovr') strategy, thus training *n_classes* models. The primal problem for LinearSVC can be formulated as equation 6.8 [69].

It makes use of the hinge loss.

$$\min_{w,b} \frac{1}{2} w^T w + C \sum_{i=1} \max(0, y_i(w^T \phi(x_i) + b)) \tag{6.8}$$

SVC, NuSVC and LinearSVC take two arrays as input, training sample X (SDR matrix of all the images) and class labels y (corresponding image). Class labels can be of string or integer type of shape *n_images*. Each of our training samples is a shape (*n_images, length_SDR*). Equation 6.9 shows the python implementation of these functions. After defining the model, we fit the input arrays to the model as shown in equation 6.10.

$$\begin{aligned} svmModel &= svm.SVC() \\ svmModel &= svm.NuSVC() \\ svmModel &= svm.LinearSVC() \end{aligned} \tag{6.9}$$

$$svmModel.fit(X, y) \tag{6.10}$$

After the training of image SDRs, we provide testing SDRs to the SVM model and predict their classes. Here, testing SDRs are SDRs of noisy images formed from trained images. This process gives us the corresponding undistorted/original image classes the distorted SDRs belong to. Equation 6.11 outputs the image classes for the test SDRs which will have the closest trained SDR.

$$classes = svmModel.predict(TestVectors) \tag{6.11}$$

6.4 Results

6.4.1 Associative Memory Results

To retrieve the shape information from noisy or incomplete data, the first step was to try a simple Associative Memory (AM). We implement the auto-associative memory using the image's component SDR (level 1) vectors without noise for training inputs and noisy image SDR vectors for testing. We calculate the weight matrix W using equation 6.1. X is the 2D matrix of shape (n, l). n is the number of training vectors and l is the length of one training vector. After calculating W in the training phase, we multiply the testing vector t to W to get the vector o. Then we use k Winner-Take-All (WTA) on vector o. k is determined such that active bits in input vector x_i and output vector y remains the same.

The basic associative memory did not work very well. We speculate that the vector space was not conducive to AM functionality, that is, similar vectors (in Hamming distance) did not necessarily represent similar shapes. We decided that re-engineering a more complex AM for our data was not worth it, so we moved to the SVM which allows more complex nonlinear partitions.

6.4.2 SVM Results

We conducted experiments to assess the ability of the SVM to learn and recognize the distorted shapes. Figure 6.10 shows 10 images which we used for training. The images are generated from the blocks world tool [21]. Each of these images is a different class.

Once training is completed, we then test on noisy versions of trained images. The noise is added in 4 ways, (a) Extra component such that it forms a connection in the graph with other components, (b) Missing component such that an existing

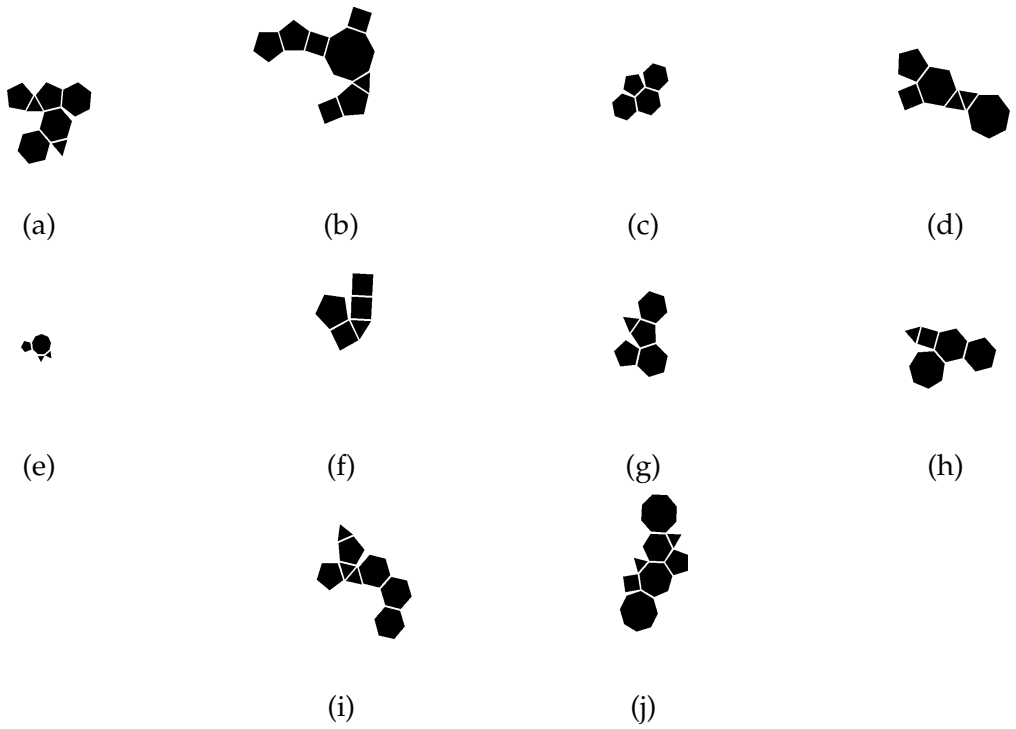
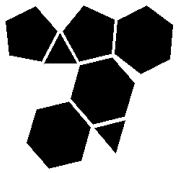


Figure 6.10: 10 Training images.

connection is broken, (c) One or two components are partially blocked, and (d) One or two components are distorted by adding noise (figure 6.11). We can group the noise into two types, (a) shape noise (extra and missing components), and (b) traditional image noise (partial occlusion and component pixel distortion). The SVM maps the SDRs of these images to the partitions of the SDR vector space it created while training original images. Based on the mapping, it returns the corresponding original / undistorted image that will have the closest SDR for the given noisy image. An image SDR stores the geometrical and structural information of shapes.

There are actually two characteristics we are trying to measure: the information the SDR captures and the efficiency of the encoding of that information. The SVM accuracy depends on both.



(a) Original image

(b) Extra component

(c) Missing component

(d) Partially blocked component

(e) Noisy component

Figure 6.11: Types of noisy versions of an image

Figure 6.12 shows 18 noisy variations of image 6.10 (b). Similarly, for all the images in Figure 6.10 used in training, we create 18 noisy variations and test the SVM model on these 180 images. We observe the classification/retrieval accuracy in two ways (a) with all the 18 variation (shape + traditional noise) and (b) with only the addition of traditional image noise, component partially blocked and distorted by noise shown in figure 6.12 (m)-(r). Table 6.1 shows the different accuracies we get from using SVC, NuSVC and LinearSVC methods.

Table 6.1: SVM model analysis for 2D blocks world images using SDRs

Images used for testing	Method	Accuracy
180 (10 classes, 18 noisy versions - shape noise + traditional image noise)	SVC	91%
	NuSVC	92%
	LinearSVC	94%
100 (10 classes, 10 noisy versions - only with the traditional image noise)	SVC	97%
	NuSVC	97%
	LinearSVC	98%

As expected, the classification is less accurate for component variations (shape noise). However, we see that the inaccurate classifications are mostly for comparatively smaller images. Smaller images are like artificial images and large/complex images are more relevant to the real world. One missing or extra component can significantly change a small image which can even puzzle humans.

6.4.3 Sparsey Results

The reason for trying Sparsey is our belief that Sparsey will modify the encoding in ways that will spread the vector space, making classification more robust leading to a more effective encoding. At the time of this writing we have not received Sparsey Results from Dr. Rinkus.

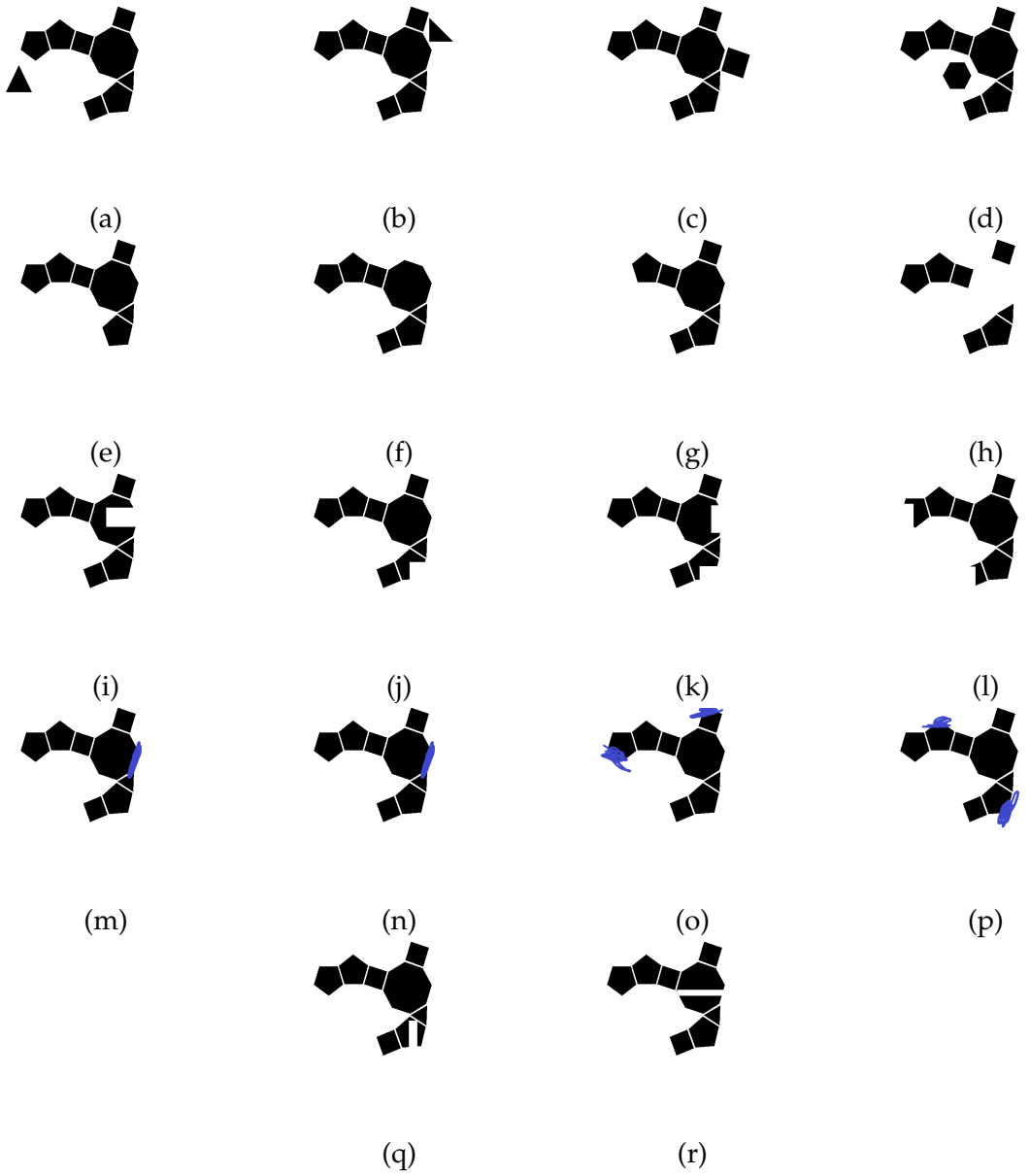


Figure 6.12: Testing images which are noisy variations of trained image 6.10(b), (a)-(d) extra component, (e)-(h) missing component / complete blocked, (i)-(l) One or two components partially blocked, (m)-(p) one or two noisy components, (q) and (r) one partially blocked component such as it is seen as two.

Chapter 7

Conclusions And Future Work

Object recognition continues to be the most important capability in computer vision. Traditional object recognition techniques were based on capturing complex features, but the features were mostly treated as unrelated in any way, the "bag of features" approach. The actual relationship of the features with respect to each other was rarely addressed, though there has been some work in this area [18], [70]. The bag of features approach loses important information about the structural relationships of the features with respect to each other, for example, the spatial relationship between the limbs of an animal or the formation and shape of vehicles. The structure captured by our SDR contains important information that may help with object recognition and complex variations of it are most likely used in primate vision. Deep networks appear to be limited in how much structure they capture. And, they are easily fooled with minor modifications of test images [2]. These failures often have to do with a common pattern in an arbitrary position a "bag of features" kind of mistake.

Graph techniques, when paired with biologically inspired representations, have the potential to be an effective method for object recognition. These techniques leverage the information about the connectedness between the features, i.e., the "structure" of an image rather than the traditional methods in which we

have no connectivity between features and objects of the image. In this thesis, we have presented a novel technique to perform object detection and pattern matching in images with the help of graph algorithms and Neuromorphic computing techniques. With these techniques, we can identify connections in images and represent those as graphs. This enables us to use many graph-based algorithms for this pattern matching in images. We showed that we can perform approximate graph matching in O(1) time with the SDR representations, and further choose k nodes subgraph in $O(n^k)$ and perform subgraph matching with $O(1)$, whereas the classic techniques take a non-polynomial amount of time. Moreover, we can also identify partial matching in images based on the inherent properties of SDRs. This work shows a way of using graph-based techniques for object recognition related tasks in images and demonstrates the use of Neuromorphic computing techniques for providing orders of magnitudes of speedups.

There are the usual sources of noise, such as speckle and partial occlusion. However, shapes also makes possible a different kind of noise: missing, extra or poorly placed components. Traditional vision algorithms generally do nor recognize shape noise. It is not clear how deep models like CNN will do in this regard. An important next step to this research is to assess that approach. The idea is to learn objects and retrieve them from any noisy or incomplete version of it. For this, we use support vector machines. The SVM partitions the image SDR vector space and maps the new test image's SDR vector to one of these vector space partitions. Therefore, it retrieves the original/uncorrupted version of test image. We trained an SVM on 10 image classes and tested on 180 images (10 classes, 18 noisy shape variations) achieving 94% accuracy. The accuracy for retained shape, but with only the addition of more traditional noise, is 98%. The classification accuracy gives us an assessment of how much information the SDR is capturing.

Future Work: The research described here has laid the ground work for a number of possible directions. Here are just a few examples.

1. Move from blocks world to more real world images. The most obvious next step is to move from blocks world to real world images. It is possible that the SDR encoding techniques will need to be expanded to accommodate real images.

2. Benchmarking against more traditional Deep Networks. Specifically it is important to compare how well traditional Deep Network models do with shape noise. This was not addressed in this thesis due to the significant effort required have the Blocks World software generate the thousands of images that Deep Models require, which put the effort beyond the scope of this thesis.

3. Studying ways to capture and represent shape using more biologically inspired networks. A major goal of our group is to understand how biological networks handle structure in data, which includes image data. Some of the specific techniques used here, such as counting the number of edges of the polygons and the relative angles of neighboring components do not appear to have direct biological equivalents. However, it is clear that biological networks capture structure, so we suspect that there are biological equivalents that roughly capture similar information.

4. Adding a probabilistic framework in graph matching. Ultimately these networks are doing probabilistic inference, adding Bayesian like measure to the shape recognition process will lead to more accurate recognition.

5. Speculating the likelihood of false positives in real applications, beyond blocks world.

6. Image SDRs can be used in big data graph analytic, which can make them faster and efficient for big database that are generally very compute intensive to process.

Bibliography

[1] A. Mathuria and D. W. Hammerstrom, "Approximate pattern matching using hierarchical graph construction and sparse distributed representation," in *Proceedings of the International Conference on Neuromorphic Systems*, ser. ICONS '19, Knoxville, TN, USA [The ACM does require the original owners/authors to obtain permission to use ACM copyrighted material]: Association for Computing Machinery, 2019, ISBN: 9781450376808. DOI: 10.1145/3354265.3354286. [Online]. Available: https://doi.org/10.1145/3354265.3354286.

[2] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, vol. 1, 2005, pp. 886–893.

[4] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, vol. 2, 1999, pp. 1150–1157.

[5] L.-J. Li, H. Su, L. Fei-Fei, and E. P. Xing, "Object bank: A high-level image representation for scene classification & semantic feature sparsification," in *Advances in neural information processing systems*, 2010, pp. 1378–1386.

[6] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International journal of computer vision*, vol. 42, no. 3, pp. 145–175, 2001.

[7] S. O'Hara and B. A. Draper, "Introduction to the bag of features paradigm for image classification and retrieval," *arXiv preprint arXiv:1101.3354*, 2011.

[8] A. Baryshnikova, "Systematic functional annotation and visualization of biological networks," *Cell systems*, vol. 2, no. 6, pp. 412–421, 2016.

[9] V. Memišević, T. Milenković, and N. Pržulj, "An integrative approach to modeling biological networks," *Journal of Integrative Bioinformatics*, vol. 7, no. 3, pp. 65–86, 2010.

[10] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[11] F. D. S. Webber, "Semantic folding theory and its application in semantic fingerprinting," *arXiv preprint arXiv:1511.08855*, 2015.

[12] J. Hawkins and S. Blakeslee, *On intelligence: How a new understanding of the brain will lead to the creation of truly intelligent machines*. Macmillan, 2007.

[13] J. Hawkins, S. Ahmad, and D. Dubinsky, "Hierarchical temporal memory including htm cortical learning algorithms," *Techical report, Numenta, Inc, Palto Alto http://www. numenta. com/htmoverview/education/HTM_CorticalLearningAlgorithms. pdf*, 2010.

[14] S. Ahmad and J. Hawkins, "Properties of sparse distributed representations and their application to hierarchical temporal memory," *arXiv preprint arXiv:1503.07469*, 2015.

[15] A. Mathuria, "Application of neuromorphic computing in object recognition," 2016.

[16] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[17] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[18] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 490–503, ISBN: 978-3-540-33839-0.

[19] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, 779–788 [The IEEE does not require individuals working on a thesis to obtain a formal reuse license].

[20] Wikipedia contributors, *Feature detection (computer vision) — Wikipedia, the free encyclopedia*, [Figure is provided according to Wikipedia Copyright License.], 2020. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Feature_detection_(computer_vision)&oldid=960964250`.

[21] D. P. Mohamed Abidalrekab, *Blocks world tool*, `https://github.com/abidalrekab/blocksWorld/tree/version3`, 2018.

[22] Y Xirouhakis, A Tirakis, and A Delopoulos, "An efficient graph representation for image retrieval based on color composition," *Advances in intelligent systems and computer science. World Scientific and Engineering Society, New York*, pp. 336–341, 1999.

[23] N. Prabhu and R Venkatesh Babu, "Attribute-graph: A graph based approach to image ranking," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1071–1079.

[24] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei, "Image retrieval using scene graphs," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3668–3678.

[25] S. Schuster, R. Krishna, A. Chang, L. Fei-Fei, and C. D. Manning, "Generating semantically precise scene graphs from textual descriptions for improved image retrieval," in *Proceedings of the fourth workshop on vision and language*, 2015, pp. 70–80.

[26] C.-W. Ngo, Y.-F. Ma, and H.-J. Zhang, "Video summarization and scene detection by graph modeling," *IEEE Transactions on circuits and systems for video technology*, vol. 15, no. 2, pp. 296–305, 2005.

[27] E. Nowak, F. Jurie, and B. Triggs, "Sampling strategies for bag-of-features image classification," in *European conference on computer vision*, Springer, 2006, pp. 490–503.

[28] A. Faheema and S. Rakshit, "Feature selection using bag-of-visual-words representation," in *2010 IEEE 2nd International Advance Computing Conference (IACC)*, IEEE, 2010, pp. 151–156.

[29] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, 2017, pp. 3856–3866.

[30] C. C. Hilgetag and A. Goulas, "
hierarchy'in the organization of brain networks," *Philosophical Transactions of the Royal Society B*, vol. 375, no. 1796, p. 20 190 319, 2020.

[31] R. Mastrandrea, A. Gabrielli, F. Piras, G. Spalletta, G. Caldarelli, and T. Gili, "Organization and hierarchy of the human functional brain network lead to a chain-like core," *Scientific reports*, vol. 7, no. 1, pp. 1–13, 2017.

[32] L. E. Suárez, R. D. Markello, R. F. Betzel, and B. Misic, "Linking structure and function in macroscale brain networks," *Trends in Cognitive Sciences*, 2020.

[33] Wikipedia contributors, *Nearest neighbor search — Wikipedia, the free encyclopedia*, [Figure is provided according to Wikipedia Copyright License.], 2020. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Nearest_neighbor_search&oldid=951092661`.

[34] O. Lézoray and L. Grady, *Image processing and analysis with graphs: theory and practice*. CRC Press, 2012.

[35] Wikipedia contributors, *Scale-invariant feature transform — Wikipedia, the free encyclopedia*, [Figure is provided according to Wikipedia Copyright License.], 2020. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Scale-invariant_feature_transform&oldid=962287797`.

[36] H. Zitouni, S. Sevil, D. Ozkan, and P. Duygulu, "Re-ranking of web image search results using a graph algorithm," in *2008 19th International Conference on Pattern Recognition*, IEEE, 2008, pp. 1–4.

[37] A. Shokoufandeh and S. Dickinson, "Graph-theoretical methods in computer vision," in *Summer School on Theoretical Aspects of Computer Science*, Springer, 2000, pp. 148–174.

[38] K. S. Camilus and V. Govindan, "A review on graph based segmentation.," *International Journal of Image, Graphics & Signal Processing*, vol. 4, no. 5, pp. 1–13,

[39] *NearestNeighbors function*, `https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors.radius_neighbors`.

[40] Wikipedia contributors, *Component (graph theory) — Wikipedia, the free encyclopedia*, [Figure is provided according to Wikipedia Copyright License.], 2019. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Component_(graph_theory)&oldid=928592532`.

[41] S. Purdy, "Encoding data for htm systems," *arXiv preprint arXiv:1602.05925*, 2016.

[42] D. George, "How the brain might work: A hierarchical and temporal model for learning and recognition," AAI3313576, PhD thesis, Stanford, CA, USA, 2008, ISBN: 9780549622079.

[43] J. Hawkins and D. George, "Hierarchical temporal memory: Concepts, theory and terminology," Technical report, Numenta, Tech. Rep., 2006.

[44] G. J. Rinkus, "Sparsey™: Event recognition via deep hierarchical sparse distributed codes," *Frontiers in computational neuroscience*, vol. 8, p. 160, 2014 [No permission needed according to the Frontiers copyright policy.]

[45] G. Rinkus, "A cortical sparse distributed coding model linking mini-and macrocolumn-scale functionality," *Frontiers in Neuroanatomy*, vol. 4, p. 17, 2010 [No permission needed according to the Frontiers copyright policy.]

[46] *Cortical.io*, `https://www.cortical.io/`, 2016.

[47] P. Kanerva, *Sparse distributed memory*. MIT press, 1988.

[48] P. J. Denning, "Sparse distributed memory," *American Scientist 77*, pp. 333–335, 1989.

[49] P. Kanerva, "Sparse distributed memory and related models," in *Associative Neural Memories*. USA: Oxford University Press, Inc., 1993, 50–76, ISBN: 0195076826.

[50] R. P. Rao and O. Fuentes, "Hierarchical learning of navigational behaviors in an autonomous robot using a predictive sparse distributed memory," *Autonomous Robots*, vol. 5, no. 3-4, pp. 297–316, 1998.

[51] B. Ratitch and D. Precup, "Sparse distributed memories for on-line value-based reinforcement learning," in *European Conference on Machine Learning*, Springer, 2004, pp. 347–358.

[52] M. Weliky, J. Fiser, R. H. Hunt, and D. N. Wagner, "Coding of natural scenes in primary visual cortex," *Neuron*, vol. 37, no. 4, pp. 703–718, 2003.

[53] T. Hromádka, M. R. DeWeese, and A. M. Zador, "Sparse representation of sounds in the unanesthetized auditory cortex," *PLoS biology*, vol. 6, no. 1, 2008.

[54] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Graph matching applications in pattern recognition and image processing," *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, vol. 2, pp. II–21, 2003.

[55] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International journal of pattern recognition and artificial intelligence*, vol. 18, no. 03, pp. 265–298, 2004.

[56] Wikipedia contributors, *Graph isomorphism — Wikipedia, the free encyclopedia*, [Figure is provided according to Wikipedia Copyright License.], 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Graph_isomorphism&oldid=951327428.

[57] Y. Zhu, L. Qin, J. X. Yu, Y. Ke, and X. Lin, "High efficiency and quality: Large graphs matching," *The VLDB Journal*, vol. 22, no. 3, pp. 345–368, 2013.

[58] A. Sanfeliu, R. Alquézar, J Andrade, J. Climent, F. Serratosa, and J Vergés, "Graph-based representations and techniques for image processing and image analysis," *Pattern recognition*, vol. 35, no. 3, pp. 639–650, 2002.

[59] *Auto-associative memory: The first step in solving cocktail party problem*, http://cs229.stanford.edu/proj2013/Youssefi-AutoassociativeMemory.pdf.

[60] J. Wetters, *Autoassociative memory*, https://www.mathworks.com/matlabcentral/fileexchange/23774-autoassociative-memory, MATLAB, 2016.

[61] Wikipedia contributors, *Support vector machine — Wikipedia, the free encyclopedia*, [Figure is provided according to Wikipedia Copyright License.], 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Support_vector_machine&oldid=962861903.

[62] 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, and E. Duchesnay, "Scikit-learn:

Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[63] *Sparsey*, `http://sparsey.com/`.

[64] K. Steinbuch and U. A. Piske, "Learning matrices and their applications," *IEEE Transactions on Electronic Computers*, no. 6, pp. 846–862, 1963.

[65] G. Palm, "On associative memory," *Biological cybernetics*, vol. 36, no. 1, pp. 19–31, 1980.

[66] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, man, and Cybernetics*, vol. 18, no. 1, pp. 49–60, 1988.

[67] S. Zhu, "Associative memory as a bayesian building block," PhD thesis, Oregon Health & Science University, Department of Science & Engineering, 2008.

[68] M. M.A. Taha, "Memristive architectures and algorithms for approximate graph-based inference," PhD thesis, Portland State University, 2020.

[69] *SVM scikit-learn mathematical formulation*, `https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation`.

[70] A. Faheema and S. Rakshit, "Feature selection using bag-of-visual-words representation," in *2010 IEEE 2nd International Advance Computing Conference (IACC)*, 2010, pp. 151–156.

Appendix A

Software and Code

IDE: PyCharm

Github code: `https://github.com/aakanksha14/ApproximatePatternMatching`

Image generation tool: `https://github.com/abidalrekab/blocksWorld/tree/version3`

Libraries:

1. NumPy - for vector and matrix handling
2. OpenCV - object detection and feature extraction
3. NetworkX and Scikit-learn - graph generation
4. Matplotlib - to plot figures