VEHICLE DETECTION USING MORPHOLOGICAL SHARED-WEIGHT NEURAL NETWORK IN THE MULTIPLE INSTANCE LEARNING FRAMEWORK

A Thesis

presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

ANES OUADOU

James Keller, Thesis Supervisor

Alina Zare, Thesis Co-supervisor

May 2017

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

VEHICLE DETECTION USING MORPHOLOGICAL SHARED-WEIGHT NEURAL NETWORK IN THE MULTIPLE INSTANCE LEARNING FRAMEWORK

presented by Anes Ouadou,

candidate for the degree of Master of Science

and hereby certify that, in their opinion, it is worthy of acceptance.

JAMES KELLER.

ALINA ZARE

MIHAIL POPESCU

ACKNOWLEDGMENTS

I would like to thank Dr. James Keller for allowing me the opportunity to work with him. I have learned a lot from him and I appreciate his guidance, patience and the time he dedicated to me as I was finding my way around my project. I am grateful to Dr. Alina Zare for jumping into the project since its first day. Her valuable advice and comments were crucial. I also would like to thank Dr. Mihail Popescu for serving on my committee and providing insight into various problems. Special thanks to my friend and partner Shuxian Shen. It was a pleasure working with you. I have learned a lot from the discussions we had throughout the execution of this project.

Finally, I thank my family for believing in me and for their continuous support

TABLE OF CONTENTS

LIST OF ILLLUSTRATIONS

Figure Page

LIST OF TABLES

Table Page

VEHICLE DETECTION USING MORPHOLOGICAL SHARED-WEIGHT NEURAL NETWORK IN THE MULTIPLE INSTANCE LEARNING FRAMEWORK

Anes Ouadou

Dr. James Keller, Thesis Supervisor

Dr. Alina Zare Thesis Co-supervisor

ABSTRACT

In this thesis, we design and implement an algorithm for object detection in aerial images based on the morphological shared-weight neural network (MSNN). The multiple instance learning (MIL) framework is used to avoid the labeling problem required in a supervised learning framework. Using the MIL, each image was given a single label. We rely on the MSNN's ability to detect objects, and on the methodology used to generate bags to find our target. Two multiple-instance MSNN structures are developed. The performance of this framework is compared with the performance of a convolutional neural network (CNN) in the same condition.

CHAPTER 1. INTRODUCTION

Supervised learning (SL) has long dominated the field of machine learning. Many applications have been developed using this approach, including computer vision applications. The use of labeled images has enabled neural networks to achieve tremendous progress. However, more complex problems require a large amount of labeled data, which is hard to obtain. Labeling, therefore, represents a serious obstacle to the progress of machine learning in the field of computer vision.

For this reason, we need an alternative approach to labeling. Multiple-instance learning (MIL) presents itself as an alternative framework with the potential to bypass this problem. Instead of labeling an object in the image, we can label the image itself. In the MIL framework, each image has only one label, which states whether the image has the targeted object (‘1’) or does not have it (‘0’). Several SL algorithms were modified and adapted to the MIL framework such as support vector machines [36], [44] and K-NN [9]. In this thesis, we use a morphological shared-weight neural network (MSNN) [4] in the MIL framework to detect vehicles in aerial images.

First, we modify the original MSNN algorithm to improve its performance, especially in terms of detection. This modification may cause the new MSNN algorithm to produce more false alarms than the original MSNN, but we believe that is worth it since the detection in the MIL framework is a tough task.

The MSNN will be adapted to use bag labels instead of instance labels. Bag generation methods are crucial in assisting the algorithms learning their tasks in the MIL framework. Different methods were developed [15], [58]. MSNN requires the instances to be in image (matrix) form to be able to use them. While this may not give a lot of room for manipulating

the images to extract the best features to be used by the MSNN; Different bag generation strategies are employed to assist the MSNN. A convolutional neural network (CNN) is the closest computer vision algorithm to the MSNN in terms of structure, optimization method, and tasks. Not to mention the high status it holds among the computer vision and the machine learning community. A CNN will also be developed for the sake of comparing its performance with the newly developed MSNN.

Finally, these algorithms' ability to detect targets in aerial images is tested. Aerial images datasets have become available and many algorithms are developed to use them in many applications such as traffic monitoring as well as security.

CHAPTER 2. BACKGROUND & RELATED WORK

2.1 Multiple Instance Learning:

2.1.1 History of Multiple Instance Learning and most important algorithms

Supervised learning has long been the primary learning framework for developing machine learning and computer vision applications. MIL is a variation of the supervised learning (SL) approach and bypasses the extensive amount of work required to prepare data for the SL approach. In SL, each data point has its own label. While this has proven useful in training diverse algorithms for applications, it is very costly in terms of time and effort, and it can be inaccurate in some cases (labeling is done manually).

Instead, MIL facilitates the labeling process. Data points (instances) are grouped in sets known as bags, and then each bag is assigned one label. In a binary classification problem, a bag can be positive, usually given label '1', or negative, usually given label '0'. A positive bag is a bag that contains at least one instance of the target, while a negative bag does not contain any. This concept is illustrated in Figure 2.1.

The first to use the term multiple instance learning was Dietterich et al. [6]. While tackling the drug activity prediction problem, Dietterich et al. tried to use the supervised learning framework to predict which drug molecule has a low energy shape that can bind with large protein molecules. A drug molecule can have hundreds of low energy shapes, but only a few of them can bind. Still, that is enough for the drug to qualify as good drug. Initially, Dietterich et al. used the supervised learning framework, labeling all the shapes of a good drug as positive examples. However, the result was unsatisfactory. Many false alarms were obtained because a drug may have hundreds of low energy shapes, but only one of them can bind. Instead, Dietterich et al. looked at it from a different perspective. A

molecule is regarded as a bag, and its different low energy shapes are the instances of that bag. So, a drug that has at least one low energy shape that can bind is considered a positive bag, while a drug that has no low energy shapes that can bind is considered a negative bag.

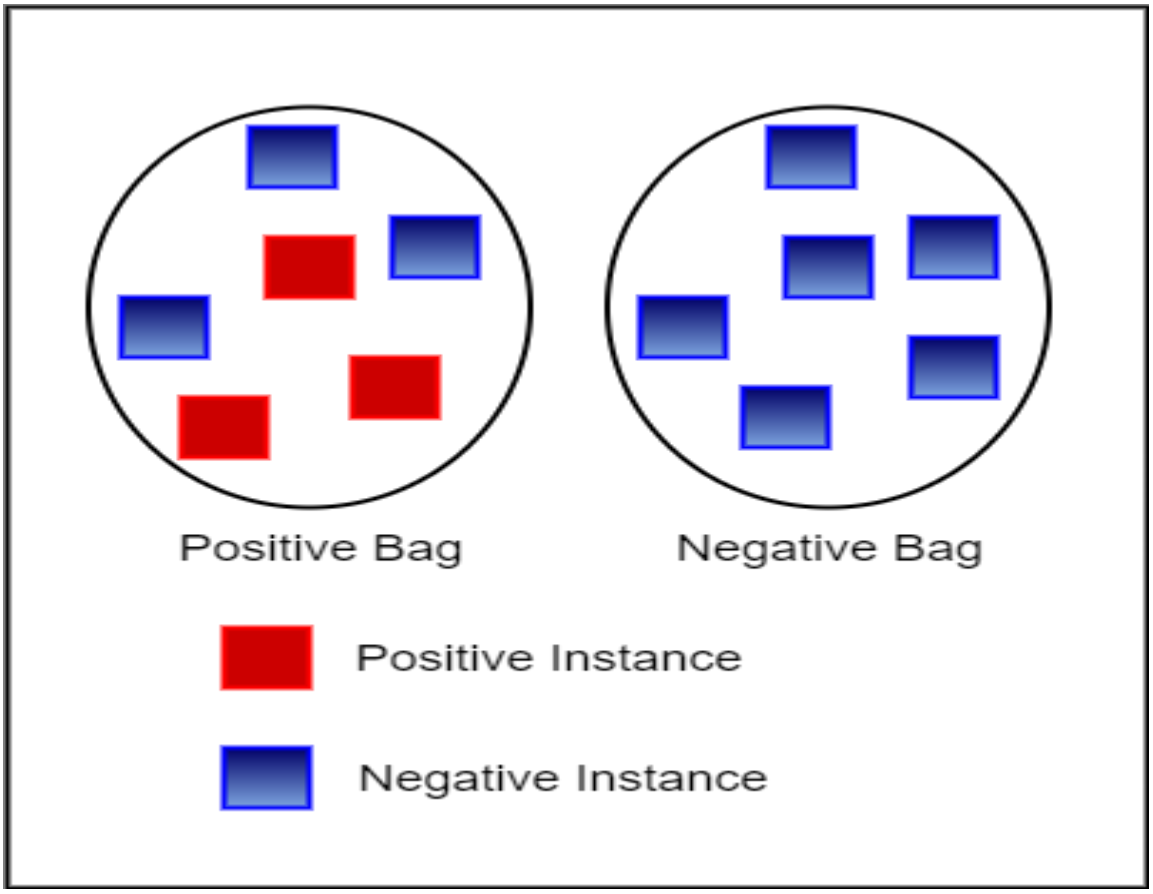


Figure 2.1 Sample of positive bag and negative bag

The establishment of MIL as a new learning framework was further confirmed by Maron and Lozano-Perez in [7]. The first attempts at developing algorithms for MIL were by Long et al. [63], Auer [45] and Blum et al. [46]; all these papers assumed that all instances from all bags are generated in an independent manner [8].

Many new algorithms were developed to be used in the new framework, such as diverse density [51], an algorithm used to measure the intersection of positive bags minus

the union of the negative bags. The purpose is to find an intersection point that can best classify the bags. This was the first paper that used MIL to learn if a person exists in an image or not, i.e. a computer vision problem.

$$\arg\underbrace{max}_{x} \prod_i \Pr(t|B_i^+) \prod_i \Pr(t|B_i^-) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.1)$$

$$t: condidate\ concept$$

$$B_i^+: ith\ positive\ bag$$

$$B_i^-: ith\ negative\ bag$$

A noisy or model is used to define the terms in Eq. (2.1)

$$\Pr(x = t|B_i^+) = 1 - \prod_j (1 - \Pr(x = t|B_{ij}^+)) \ldots\ldots\ldots\ldots\ldots\ldots \quad (2.2)$$

$$\Pr(x = t|B_i^-) = \prod_j (1 - \Pr(x = t|B_{ij}^-)) \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.3)$$

The probability of an individual instance is defined in Eq. (2.4)

$$\Pr\left(x = t\middle|B_{ij}\right) = \exp(-\left\|B_{ij} - x\right\|^2) \ldots\ldots\ldots\ldots \quad (2.4)$$

Other experts tried to adapt existing algorithms used in supervised learning to MIL. Jun Wang et al. [9] adapted the K nearest neighbor method and came up with two new algorithms, called Bayesian-KNN and Citation-KNN. The initial approach was to use the Hausdorff distance to measure the distance between the bags, but this method led to a contradiction referred to by [9] as the "minor being winner". The K-NN is non-parametric method that classifies a test point according to the majority vote of its neighbors. The same process is applied except with bags. Due to the way the bags are defined, there is a possibility that the correct label of the test bag is the label of the class with the smaller

number of examples, which contradicts the definition of the K-NN. For instance, a test bag can have more negative bags around it than positive, so the label assigned is negative. But the correct label is positive. This is illustrated in Figure2.2. To solve this problem, alternative approaches were used. The first one was the Bayesian approach. In this approach, the explicit probability of each hypothesis is calculated and used to classify the test bag, instead of counting the number of bags, then classify according to the class that has the majority of bags around the test bag. The other method is called the citation approach. This method is based on the idea of how documents function as a reference (cited by another document) or citer (cites other documents as reference). Projecting the idea on our case, for example, we consider both the bag's neighbors and the bags that consider this bag as a neighbor to classify it. By comparing the number of positive and negative bags in both the reference and citer, the bag's class is decided. Y. Chevaleyre and J.-D. Zucker [10] used MIL to solve the multiple part problem. A standard decision tree was changed to a multiple decision tree to fit the multiple instance framework.

Dietterich stated that neural networks are not fit to be used in the MIL framework. ML Zhang et al attempted to overcome this issue [11] by proposing a neural network for MIL. Their main contribution was a new error function: since each bag is made up of multiple instances, the new function will back-propagate the instance producing the maximum error. The performance of this algorithm is comparable to other MIL methods. In [12], ML Zhang et al came up with an improvement of their previous algorithm by including preprocessing of the feature vectors before they are fed to the neural networks. They proposed two methods for that: PCA and Diversity Density.

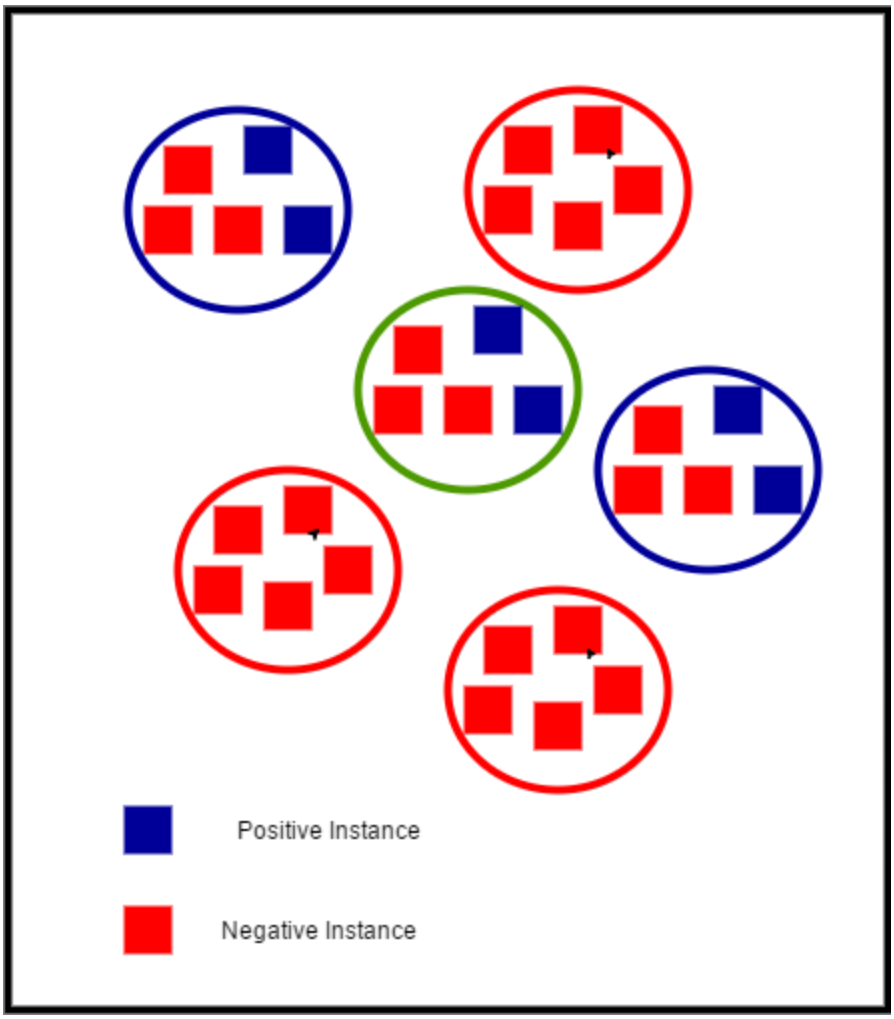


Figure 2.2 Multiple Instance Learning K-NN. Blue circle is a positive bag. Red circle is a negative bag. Green circle is a test bag. When Hausdorff distance is used, the test bag is classified negative, but the correct classification is positive. This is what is referred to as "minor being winner"

S. Andrews et al [36] proposed two methods to extend the definition of SVM to the MIL framework. The two methods are called mi-SVM also known as "the Maximum Patten Margin Formulation of MIL", and MI-SVM also known as "the Maximum Bag Margin Formulation of MIL". The mi-SVM differs from the SL case by not knowing the actual label of the individual patterns. If the pattern belongs to a negative bag then it is definitely negative, whereas if it belongs to a positive bag then there is a chance that it could be positive. Therefore, this pattern is treated as an unknown integer variable. This

approach is trying to find a hyperplane that classifies all the negative instances of the negative bag as negative and at least one instance of the positive bag as positive. In the MI-SVM case the focus is on classifying the bag correctly. A positive bag is represented by its most positive pattern, while a negative bag is represented by its least negative pattern. Once these two patterns are identified for each bag, then the remaining patterns are irrelevant. For mi-SVM case, the integer variable is a part of the optimization process. For the MI-SVM case, a selector variable is used to determine the positive pattern that will represent the bag. Both methods involve the use of integers which complicates the optimization process. Heuristics are developed to handle these problems.

$$mi - SVM \underbrace{min}_{\{y_i\}} \underbrace{min}_{w,b,\xi} \frac{1}{2} \|W\|^2 + C \sum_i \xi_i \quad \text{..... (2.5)}$$

$$s.t. \forall i : \; y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i, \xi_i \geq 0, y_i \in \{-1,1\}$$

$$MI - SVM \underbrace{min}_{w,b,\xi} \frac{1}{2} \|W\|^2 + C \sum_I \xi_I \quad \text{..... (2.6)}$$

$$s.t. \forall I : \; Y_I \underbrace{max}_{i \in I} (\langle w, x_i \rangle + b) \geq 1 - \xi_i, \xi_i \geq 0$$

Babenko et all [37], [38] used the MIL to solve the problem of drift in tracking systems. An online MIL version of AdaBoost is also proposed. This tracking system updates the appearance model, then applies it to a window around the last known location of the target. Finally, the tracking state is updated. This process is repeated for each frame. The appearance model is based on a discriminative binary classifier that classifies image patches as target or non-target. It is built using the AdaBooting method. The frames are

represented using the Haar features. In traditional tracking systems, one patch is selected where the target is likely to be. The problem with this approach is that the system may pick a good patch, but not the optimal one. The system will slowly drift from the target. In the proposed system, when a frame is received, the last recorded location is used to randomly pick image patches around that area within a radius, S. Negative instances are image patches cropped at distance $S < \cdots < B$. Each of these instances is a negative bag by itself. These bags are used to train the classifier, i.e. update the appearance model

Zhang et al [39] used Expectation Maximization (EM) to improve the performance of Diverse Density (DD) [51]. By introducing EM, the MIL problem is converted into a SL problem. DD uses gradient search that finds a hypothesis h that maximizes $DD(h)$. A first random point is chosen, then a two-step process is repeated until the best hypothesis is found. The first step is the Expectation Step. The current hypothesis h is used to pick one instance from the bag that is most likely responsible for the bag label. The second step is called the Maximization Step. DD is used to find a new h' that maximizes $DD(h)$. h is replaced by h' and the process is repeated. Using this approach, the problem is converted into the SL framework.1-norm SVM [50] is used to select the important features and eliminate the redundant features. A 1-norm SVM classifier is constructed simultaneously. We can see the difference between 1-norm SVM and quadratic SVM in Eq. (2.7) and Eq. (2.8), respectively.

$$1-norm\ SVM\text{:}\ \underbrace{min}_{w}\ [1 - y_i(b + w^T m)] + \lambda\|w\|_1 \text{..... (2.7)}$$

$$quadratic\ SVM\text{:}\ \underbrace{min}_{w}\ [1 - y_i(b + w^T m)] + \lambda\|w\|_2^2 \text{..... (2.8)}$$

The final number of features is determined by the selection process based on the 1-nom SVM. This is done by introducing two penalties, one for positive bags C_1 and one for negative bags C_2 , this is shown in Eq. (2.9)

$$\underbrace{min}_{w,b,\xi,\eta} \lambda \sum_{k=1}^{n}|w_k| + C_1 \sum_{i=1}^{l^+} \xi_i + C_1 \sum_{j=1}^{l^-} \eta_i \text{..... (2.9)}$$

$$\text{s.t. } (w^T m_i^+ + b) + \xi_i \geq 1, i = 1, \dots, l^+,$$

$$-(w^T m_i^- + b) + \eta_j \geq 1, j = 1, \dots, l^-,$$

$$\xi_i, \eta_j \geq 0, i = 1, \dots, l^+, j = 1, \dots, l^-,$$

Where ξ_i, η_j are hinge losses. The values for C_1 and C_2need to be chosen in way that does not favor one over the other. So, let's have $C_1 = \mu$, then $C_2 = 1 - \mu$ where $0 < \mu < 1$. To form a linear program,

$w_k = \mu_k - v_k$ where $\mu_k, v_k > 0$. Since $\mu_k$or v_k must equal 0, then

$$|w_k| = \mu_k + v_k$$

then Eq. (2.9) is rewritten as

$$\underbrace{min}_{w,b,\xi,\eta} \lambda \sum_{k=1}^{n}(\mu_k + v_k) + \mu \sum_{i=1}^{l^+} \xi_i + (1-\mu) \sum_{j=1}^{l^-} \eta_i \text{..... (2.10)}$$

$$\text{s.t. } ((\mu - v)^T m_i^+ + b) + \xi_i \geq 1, i = 1, \dots, l^+,$$

$$-((\mu - v)^T m_i^- + b) + \eta_j \geq 1, j = 1, \dots, l^-,$$

$$\mu_k, v_k > 0, k = 1, \dots, n$$

$$\xi_i, \eta_j \geq 0, i = 1, \dots, l^+, j = 1, \dots, l^-,$$

Assuming $w^* = \mu^* - v^*$ and $b^*$is the optimal solution for Eq. (2.10), the influence of the kth feature is determined by the magnitude of w_k^*. Only the features with nonzero w_k^* are selected.

This method is considered a "wrapper method" because the final classifier is also involved in the feature selection process. This process is computationally expensive, which is why 1-norm SVM is used instead of the quadratic SVM. This obtained classifier can classify bags. To classify instances, further work is needed. The contribution of the instances to the classification of a bag is used as a reference. A threshold is then defined; if the instance contribution is higher than the threshold then the instance is classified positive. If the contribution is less than the threshold, then the instance is classified negative.

Maron et al. [15] explored the use of MIL in the field of natural scene classification. They also investigated the effect of using various bag generators on the final outcome if a simple algorithm such diverse density is used. Several bag generators were tested on RGB colored images. The main idea of these generators is to try to get as much descriptive information as possible by exploring the relationship between the pixel and its neighbors. The information obtained are arranged in vectors. Each vector is considered an instance.

Ali et al. [41] used the MIL for the recognition of human action in videos. Kinematic features were used for recognition. These features were obtained from the optical flow of the video. Principal component analysis (PCA) is applied to the spatiotemporal volume of features. Spatiotemporal patterns are obtained from the video sequences. A key part of the proposed method is formulating the bags. For each bag, a video of action is extracted. Optical flow between consecutive frames is computed and stacked. Kinematic features are computed and a spatiotemporal volume is produced for

each feature. The next step is to produce kinematic modes; this is done by applying PCA to the volumes of each feature. These kinematic modes are the instances of the bag. The created bag is used for the embedding of the video in the feature space. Classification is then done using the K-NN.

Zhang et al. [42] used the MIL for content-based image retrieval (CBIR). CBIR deals with how an image is systematically retrieved from a large dataset. A comparative study is performed when different factors may influence a given CBIR problem. Two algorithms are compared in this study: distributive density (DD) and expectation maximization-diverse density (EM-DD). Each image is considered as a bag. The instances are chosen using segmentation based on two approaches. In the first approach the image is smoothed and then segmented using a fixed scheme. In the other approach the image is segmented at its original resolution; then, one instance is taken from each segment. Each of these instances are converted into a feature vector, using a variety of methods. Finally, two systems are employed for representing colored images. The RGB system and the luminance-chrominate, also known as YCrCb, were both used.

Ray et al. [43] compared the performance of some SL algorithms with their counterparts in MIL. Four algorithms that represent different approaches were tested in SL and MIL frameworks. Different datasets from different applications were used. The algorithms used were diverse density (DD) in MIL versus a simple Gaussian model of DD in SL. The SL DD is created by replacing Eq. (2.2) with Eq. (2.11). while everything else remained the same. Normalized set kernel SVM in MIL was compared to quadratic SVM in SL, and MIL first order inductive learner (FOIL) was compared with SL FOIL. Finally, MIL linear regression was compared with SL linear regression. Four datasets representing

four different applications were used for testing: drug activity, CBIR, protein identification and text categorization. The performance was evaluated using area under the curve of ROC curves from 10-fold cross validation. To compare the results obtained from the SL level with the MIL results on equal terms, the confidences of all the instances from one bag were compared and the highest confidence was considered as the one representing the bag.

$$\Pr(x = t|B_i^+) = \prod_j (\Pr(x = t|B_{ij}^+)) \ldots\ldots\ldots\ldots\ldots (2.11)$$

For the Musk datasets (drug activity), MIL algorithms were always superior to their SL counterparts except for FOIL. DD -an algorithm originally designed to tackle MIL problems- had a better performance than its SL counterpart in all categories. While the inverse happened for FOIL, the SL version was always superior to the MIL version. For the SVM, the MIL version had better performance with the Musk dataset, while SL version had better performance in the text categorization. For CBIR and protein identification, their performance was very close with no clear winner. Finally, MIL linear regression achieved a better score than SL linear regression in Musk dataset, protein identification dataset and CBIR. The difference between the two algorithms is small in the CBIR dataset. For the text categorization, there is not a clear winner.

2.1.2 Computer vision applications and Multiple Instance Learning

Multiple Instance Learning (MIL) was extensively used in computer vision applications. The first obvious application of MIL in computer vision is content based image retrieval (CBIR) [15], [42], [43]. This problem was for a long time the most studied, it focuses on classifying images based on what they contain. In his application, the location of the target is not important. Different bag generation methods were tested and investigated [15], [58]. In CBIR images need to be partitioned and feature vectors are

created. There are different ways of partitioning the images, Yang et al. [59] used semantic regions, Maron et al. [15] used a system that looks like a grid, and Csurka et al. [60] used bag of key points.

The other MIL application is object localization and image segmentation. Given the weak labeling of an image, MIL is found to be a good alternative for SL which required the labeling of all the data. MIL is used to localize a target(s) [37], [38], [54], and to segment an image [55]. MIL was found suitable for detection in videos, where the target often appears in many frames at different locations, which makes the labeling hard [37], [38], [61] or when trying to detect an action that spans over several frames in a video [41] and [62].

2.1.3 Multiple Instance Learning and convolutional neural network

Like other computer vision algorithms, MIL found its way to the CNN. MIL is used to alleviate some of the problems CNN faces during the training. One of the problems in which MIL is used is data augmentation. Papandreau et al. [52] designed a CNN that uses epitome images instead of the convolution-polling layers. Each input image was augmented into a set of images. These images are subjected to a scale-translation transform. The purpose of doing this is to allow the CNN to learn targets at different locations and scales. Since MIL is used, the loss function must be changed accordingly. There are different methods to do that. In [52] the loss function was updated in a way that allows the classifier to choose a preferred transform.

$$L(y_i, X_i) = l(y_i, \underbrace{\max_k f(X_i^k)}) \quad \text{..... (2.12)}$$

Where,

$$X_i: the\ ith\ input\ image$$

$$y_i: the\ ith\ label\ of\ the\ ith\ input\ image$$

$$X_i^k: the\ kth\ transform\ of\ the\ ith\ input\ image$$

$$f(.): the\ classifier$$

Sun et al. [53] also used the MIL for data augmentation. However, this time MIL was used to address the problem of data shortage, that is, to create more data to train the CNN. The use of MIL promoted a modification to the CNN, in which the SoftMax function usually found in the output layer of a CNN has been replaced by MIL layer. This layer is defined according to Eq. (2.13)

$$P\left(c_i = 1 \middle| X^j\right) = 1 - \exp(-\lambda h_i^j) \quad \text{..... (2.13)}$$

Where,

$$c_i: the\ ith\ category$$

$$X^j: the\ jth\ instance\ of\ a\ bag$$

$$h_i^j: the\ ith\ output\ of\ the\ CNN\ model\ before\ loss\ layer\ for\ jth\ region$$

$$\lambda: constant\ positive\ value$$

Cinbis et al. [54] used what is referred to as the standard MIL training method to train a CNN. In this method, the highest scoring detection is considered the positive training example and it is used to train the detection model. The detection window is refined using an edge-driven objectness measure [57]. Using this approach, a detection window

was scored according to how many complete contours it contains. The more it has the higher the score.

Pinheiro et al. [55] designed a pixel level segmentation algorithm based on CNN. During training, CNN produced |C|+1 planes, where C is the number of classes and the extra plane is dedicated for the background. The pixels' scores are aggregated using Eq. (2.14). Then, MIL is used to compute the image-level scores using the planes scores as shown in Eq. (2.15). the image level label is treated as a conditional ability and SoftMax is used to compute it.

$$S^k = \frac{1}{r}\log[\frac{1}{h^0 w^0}\sum_{i,j}\exp(r, S_{i,j}^k)] \quad \text{(2.14)}$$

$$p(K|I,\theta) = \frac{exp(S^k)}{\sum_{c\in C}\exp(S^k)} \quad \text{(2.15)}$$

During the test, the aggregation layer is removed, and the obtained result is the segmentation of the image.

Yang et al. [56] combined the weak labels of MIL with the strong labels to solve the multi-class problem. A new framework is created and named: multi-view multi-instance framework. First, multi-class problem is transformed into an MIL problem by generating object proposals from each image, each image is treated as a bag. Next, generate feature view for each object proposal using a pre-trained CNN. The strong labels are used to establish neighborhood relationship between the different regions.

2.2 Morphological Shared Weight Neural Network

2.2.1 Introduction:

The concept of a "neural network" started with the introduction of Rosenblatt's perceptron and its convergence theorem [47]. But the interest in neural networks quickly faded when Minsky and Papert demonstrated the limits of using the perceptron [48]. The concept of a neural network did not regain its position even with the development of back-propagation by Werbos in his Ph.D. thesis [49]. It was not until 1986, when the book "Parallel Distributed Processing: Explorations in the Microstructures of Cognition" was published [40], that the neural network came back into prominence.

An efficient feature extractor is crucial to the performance of a neural network. A system is usually made up of two phases. The first phase is the feature extraction; data is transformed into low dimensional feature vectors that can provide an invariant description of the data. The second phase consists of a classifier that is trained using the feature vectors.

In order to improve the preprocessing, a layer was introduced before the neural network. Instead of extracting a set of features for a task using a feature extractor, this added layer would learn the features suitable for that task, along with the weights and biases of the neural network. An important milestone was the introduction of the "convolution layer." One of the first successful networks was LeNet, developed by LeCun [1]. This network used three types of layers: convolutional, pooling, and fully connected. It was successfully used for the classification of handwritten digits.

The success of the convolutional neural network remained unnoticed and slow progress was made in its development until the year 2012, when the Dan Ciresan Net [2] was introduced. This network was among the first to take advantage of the use of GPU in

its design. In the same year, AlexNet was introduced [3]. It is a deeper and wider version of LeNet. It also uses GPUs in its training. A new activation function known as ReLU, which stands for Rectified Linear Unit, was also introduced. AlexNet started a revolution in the field of neural networks, and ReLU avoided the saturation problem common with sigmoid function.

Around the time LeNet was introduced, another algorithm appeared, but it didn't gain much attention at the time. The morphological shared-weight neural network was presented by Won in his Ph.D. dissertation [(4)]. It followed the same idea as the convolutional neural network in terms of learning the features using a newly added layer. The main difference is that, while the convolutional neural network uses convolution operation in its feature learning layer, while MSNN uses morphological operations for the feature learning layer.

2.2.2 Morphological Shared Weight Neural Network

Morphology is one of the basic operations in image processing. It focuses on the shapes that exist in an image and their geometry. The basic operations are "Dilation" and "Erosion." Depending on how these two operations are combined, more complex operations can be created such as "Opening," which is erosion followed by dilation using the same structuring element for both operations. "Closing" is the reverse of "Opening," that is, dilation followed by erosion using the same structuring element for both operations. Morphological operations are used in different image processing tasks such as noise elimination, segmentation, and image reconstruction.

Morphological operations were originally performed in the binary domain, but they can be extended to the grayscale domain.

2.2.3 MSNN

The MSNN is composed of three types of layers: morphological, pooling and fully connected. Figure 2.2 shows a diagram for the MSNN. We find the fully connected layer in the neural network part of the MSNN. The morphological layer is where the features are learned, while the pooling layer serves to reduce the dimensionality of the feature maps. Each morphological layer has at least one pair of structuring elements, the hit and miss. Using the hit-miss transform one feature map is produced. The hit is based on the erosion operation while the miss is based on the dilation operation.

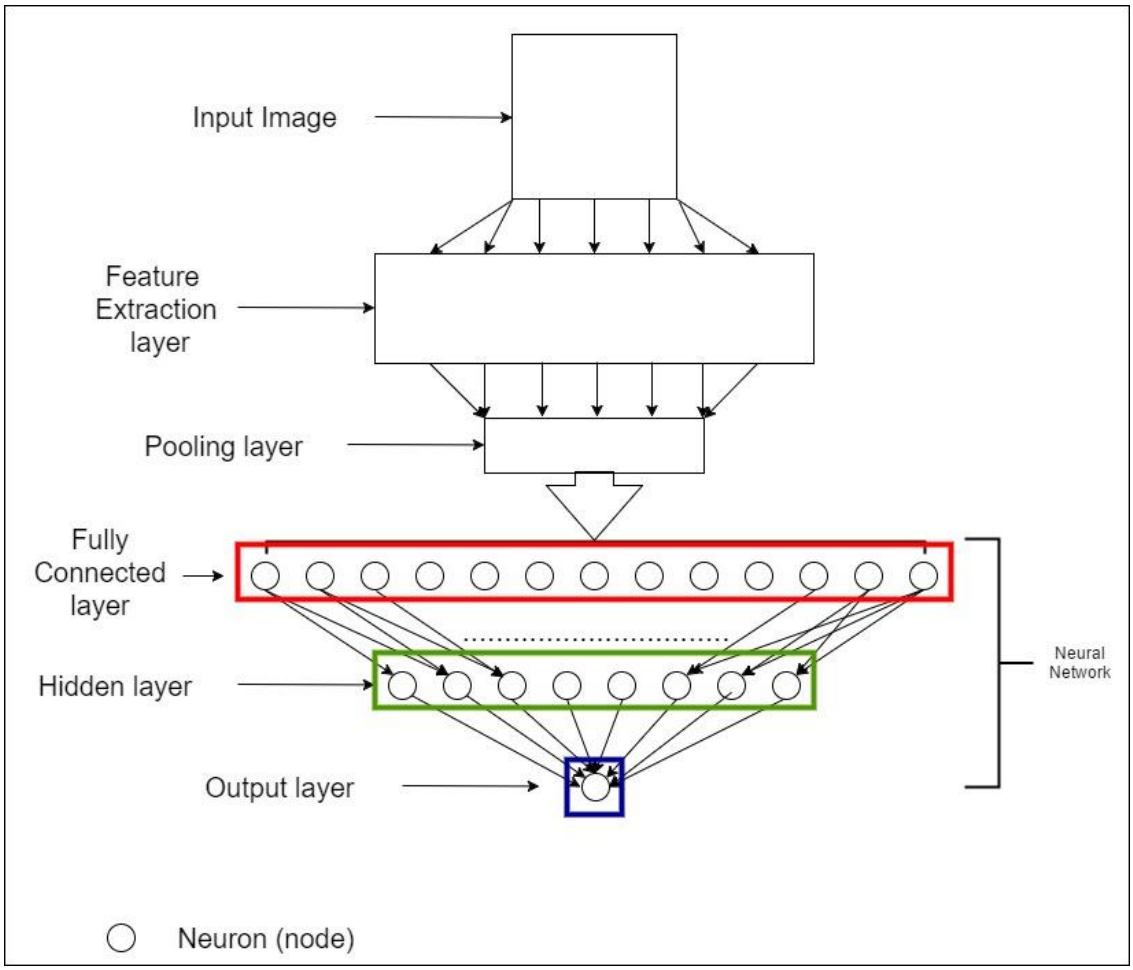


Figure 2.3 Diagram of an MSNN network with one feature extraction layer and one pooling layer. The neural network layers are the fully connected in red, the hidden in green, and the output layer in blue

A) Binary morphological operation

Binary Erosion:

Erosion is one of the basic morphological operations. According to [4], it is used to identify the points at which a structuring element fits inside the image. It is dual to the dilation.

We can use the set theory to explain the relationship between an image and a structuring element. According to [5]:

If A and B denote two sets with elements a and b, respectively. Then, the *binary erosion* of A and B is the set of all elements x, for which $x + b \in A$ for every $b \in B$.

So the binary erosion of A by B, denoted by $A \ominus B$, can be defined as follows:

$$E(A,B) = A \ominus B = \{x: B + x \subset A\} \quad \text{(2.16)}$$

Figure 2.4 shows an example of the erosion operation in a binary image. Figure 2.4(a) is the original image, Figure 2.4(b) is the structuring element, and Figure 2.4(c) is the result of the erosion. We can see how the image was eroded. When performing the erosion operation, the structuring element is translated over the image. The pixel being evaluated is the one on top of the center of the structuring element. This pixel is deemed a part of the resulting image if all the elements of the structuring element are inside the image; if one of the pixels is outside the image, then that pixel is eroded.

In this definition, A represents the input image and B is the structuring element. We can rewrite the definition in a form that fits image processing, that is, the intersection of two images.

$$E(A,B) = A \ominus B = \cap \{A - b: b \in B\} \quad \text{(2.17)}$$

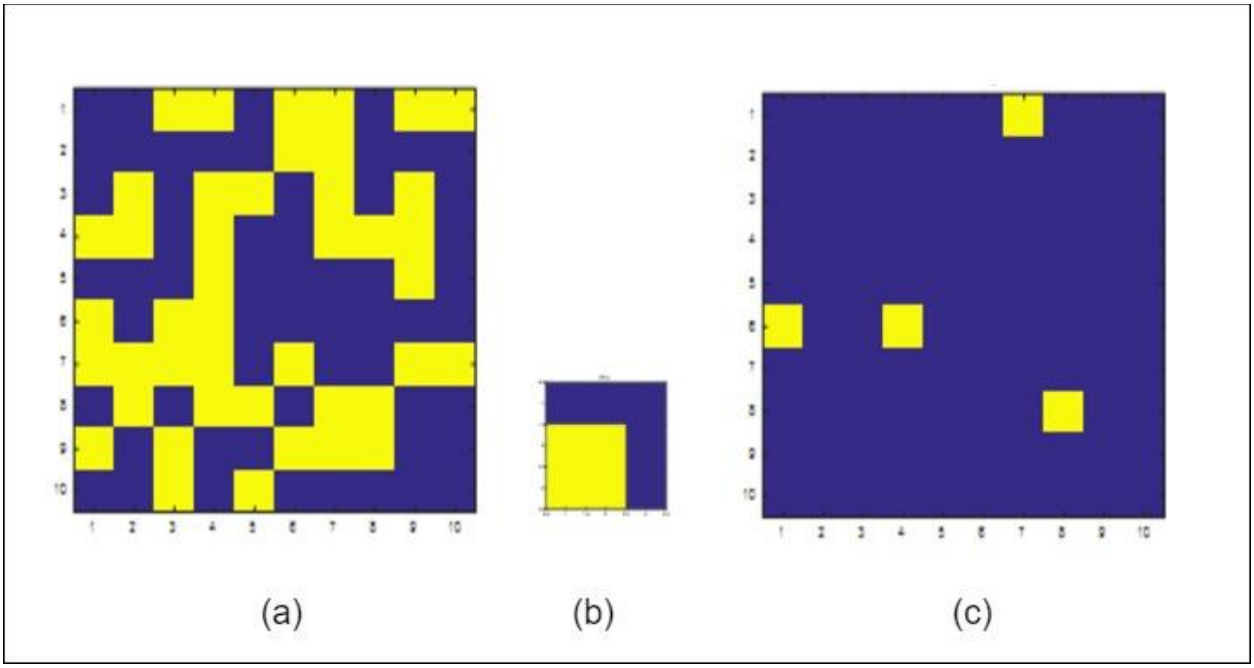


Figure 2.4 Example of binary Erosion. (a) the original binary image.
(b) the Structuring Element, (c)the resulting image.

Binary Dilation:

Binary dilation is the other basic morphological operation. It is dual to erosion; therefore, we can use erosion and set theory to define the dilation [4]:

$$D(A,B) = A \oplus B = [A^c \ominus (-B)]^c \ldots\ldots\ldots\ldots\ldots\ldots (2.18)$$

Dilation can also be written using Minkowski addition; this formula suits better the image processing context:

$$D(A,B) = A \oplus B = \cup \{A + b : b \in B\} \ldots\ldots\ldots\ldots\ldots\ldots (2.19)$$

Figure 2.5 shows an example of dilation: Figure 2.5(a) is the original image, Figure 2.5(b) is the structuring element, and Figure 2.5(c) shows the result of the dilation. We can see how the image was dilated. When performing the dilation operation, the structuring

element is translated over the image; the pixel being evaluated is the pixel with the structuring element center on top of it. If at least one of the elements of the structuring element is still on the image, then that pixel is considered a part of the image and the image is, therefore, dilated

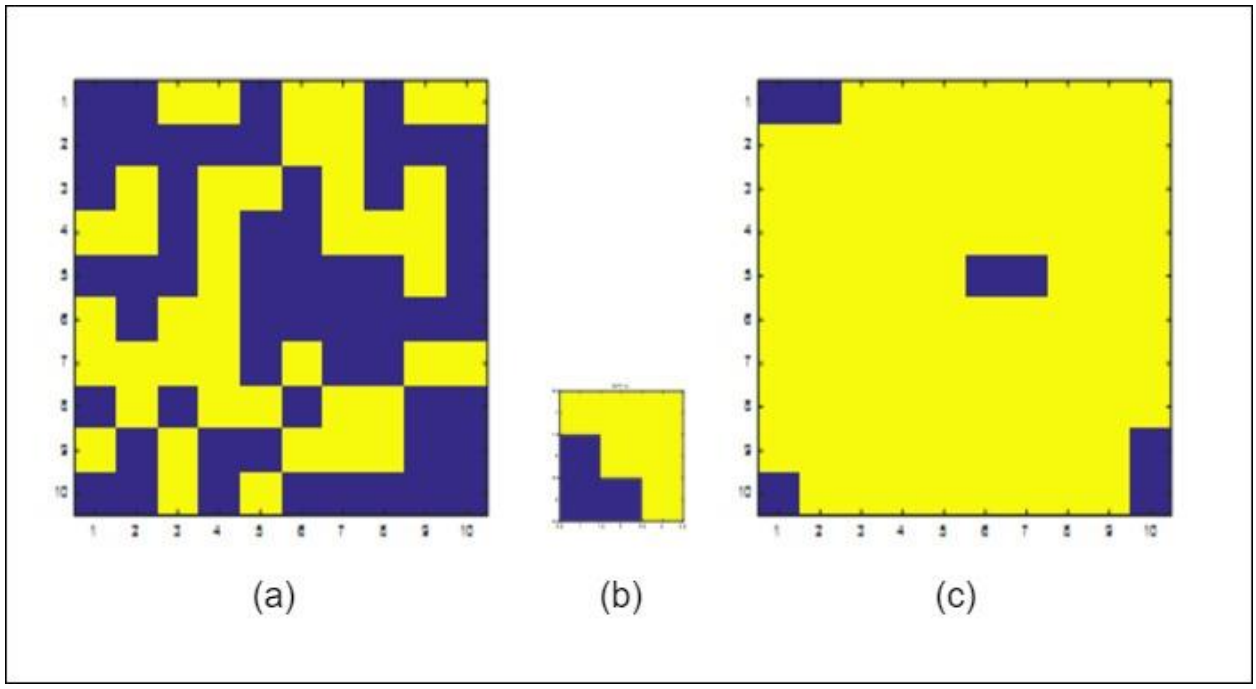


Figure 2.5: Example of binary dilation. (a) The original image, (b) the structuring element (SE), and (c) the resulting image

Binary Hit-Miss Transform

This is one of the operations that uses a combination of erosion and dilation operations to produce a specific effect. It is defined [4] as follows:

$$A \otimes B = A \otimes (B_1, B_2) = (A \ominus B_1) \cap (A^c \oplus B_2) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.20)$$

where B_1 and B_2 are the structuring elements. The first part of this operation probes the inside of the image while the second part probes the outside of the image. Then, the

difference between the two parts is measured. Figure 2.6 shows the result of the Hit-Miss operation.

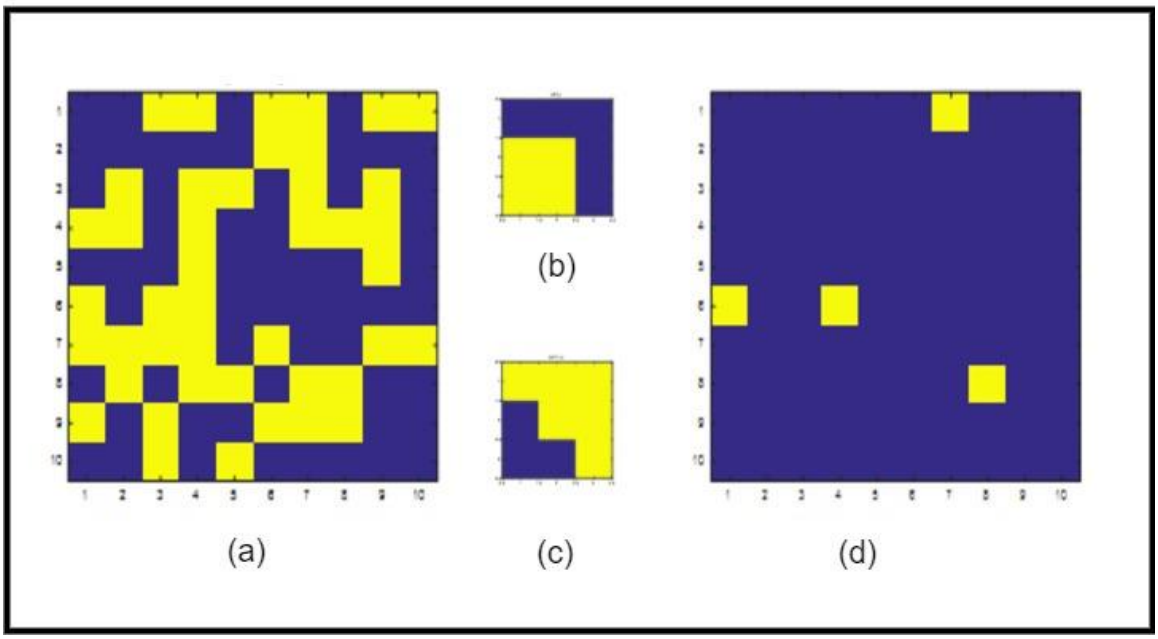


Figure 2.6 Binary hit-miss transform. (a) the original image. (b) the hit structuring element, (c) the miss structuring element, (d) the result of the hit-miss transofirm

B) Grayscale morphological operations

In computer vision applications, images are either colored or grayscale. Our MSNN operates on grayscale images. Therefore, the current definitions of the 'hit' and 'miss' are not practical and they need to be extended to grayscale. Won [4] used the umbra transform to achieve that. While the binary morphology is defined by set theory, the umbra transform enables us to look at a grayscale image as a three-dimensional binary image.

Let's consider the function $f: X \to R$ where X is a subset of n-dimensional Euclidean space. This function represents the grayscale image. "*the umbra of a function f is the set of all the points in the space beneath 'f' including the points on the function itself and is defined as*" [4]

$$U[f] = \{(x, y): x \in D[f] and\ y \leq f(x) where\ y \in R\} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.21)$$

D[f] is the domain of the function, where $D[f] \subset X \times R$.

Let's consider the subset $A \subset X \times R$. The surface for set 'A' is defined as follows:

$$S[A] = \{(x, y) \in A: y \geq z\ for\ any\ (x, z) \in A\} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.22)$$

the surface acts as the inverse of the umbra transform.

All the properties used by binary morphology are also available in the grayscale morphology including intersection, union, complementation and translation. To put it in a context more appropriate to the discrete grayscale image let's define our image as:

$$I = f(x)$$

where $'I'$ is the intensity at the coordinates x.

Our structuring element can also be defined in the same way as:

$$i = SE(x)\ .$$

Using the umbra transform, we get:

$$U(f) = \{(xz) |\ x \in D_f\ and\ z \leq f(x)\ \} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.23)$$

where D_f is the domain of f .

Using this definition of the umbra transform, we can define our erosion and dilation as follows:

Erosion:

$$(f \ominus SE)(x) = \min \{f(y) - SE(y - x) |\ y \in D_{SE}\} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.24)$$

Dilation:

$$(f \oplus SE)(x) = \max\{f(y) - SE(y - x) | y \in D_{SE}\} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.25)$$

The erosion and dilation in grayscale use the min and max, respectively. 'Min' is equivalent to intersection and 'Max' is equivalent to union.

Won [4] defined the grayscale hit-miss transform [64] as:

$$f \otimes (h, m) = (f \ominus h) - (f \oplus m) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.26)$$

where $'h'$ and $'m'$ are the structuring elements for the 'hit' and 'miss,' respectively. According to Won [4]

> "… *if the structuring elements are chosen to match the shape of a given grayscale image over a region I, then the hit-miss Transform as defined above will produce a peak whenever that shape occurs in the input*." --Y. Won, 1995, p. 56

As shown in Figure 2.2, MSNN is made up of two parts. Part one is where the features are learned, while part two is a standard neural network. Figure 2.7 shows the schematic of the feature extraction layer. Details of the operations performed in this layer are explained below

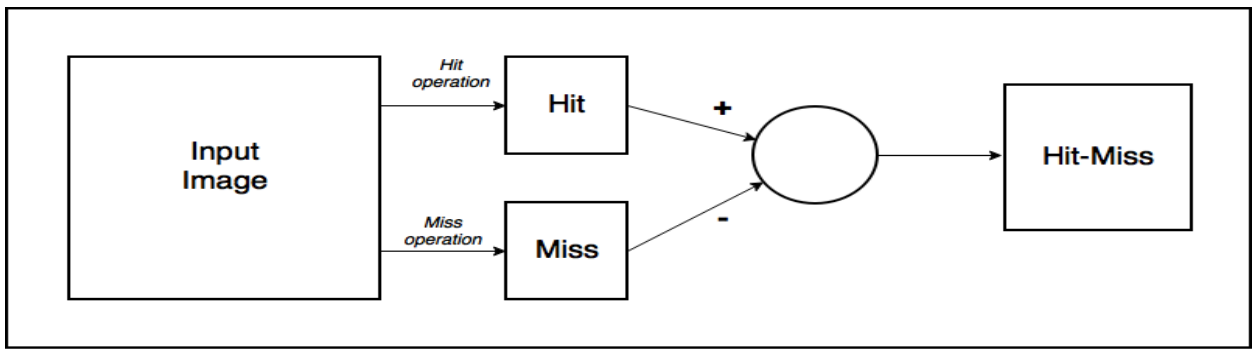


Figure 2.7 Diagram of the featured extraction layer

C) MSNN feature extraction layer:

In the feature extraction layer, the three operations, hit, miss, and hit-miss, are performed. Figure 2.8 shows how These three operations are performed at the node level.

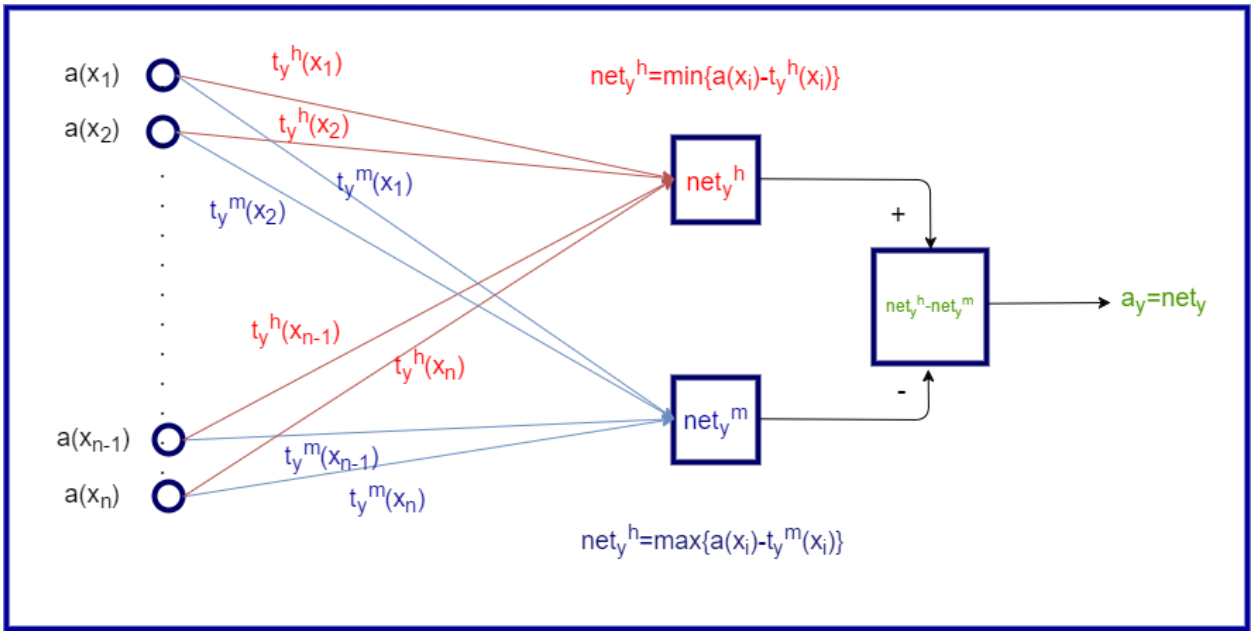


Figure.2.8 Node operation in feature extraction layer. Hit operation is displayed in red. Miss operation is displayed in blue. The output of the hit-miss transform is shown in green.

The following nomenclature will help the reader interpret the equations needed to understand this process.

$$a(x): the\ input\ to\ a\ node$$

$$t_y^h(x): the\ structuring\ element\ for\ the\ Hit\ operation$$

$$t_y^m(x): the\ structuring\ element\ for\ the\ Miss\ operation$$

$$net_y^h(x): the\ output\ of\ the\ hit\ operation$$

$$net_y^m(x): the\ output\ of\ the\ miss\ operation$$

$$a_y: output\ of\ the\ hit-miss\ transform$$

the forward pass:

the hit $$net_y^h = \underbrace{min}_{x \in D[t_y]} \{a(x) - t_y^h(x)\} \text{.....(2.27)}$$

the miss $$net_y^m = \underbrace{max}_{x \in D[t_y]} \{a(x) - t_y^m(x)\} \text{..... (2.28)}$$

the output of the node is:

$$a_y = net_y^h - net_y^m \text{..... (2.29)}$$

A sample of hit and miss operations are illustrated in Figure 2.9 and 2.10, respectively. The structuring element (SE) slides one pixel at a time. The part of the image occupied by the SE at each step is '$a(x)$'. For each position, Eq. (2.27) is performed. The steps of Eq. (2.27) for the hit are displayed in Figure 2.8. In the first column, we 5x5 matrix representing the input image. In the next column, we have a red matrix representing an area of the input image equal to the size of the SE. This area is equivalent to '$a(x)$' in Eq. (2.27). In the next column, we have the SE in light blue $'t_y^h'$. The result of the subtraction $\{a(x) - t_y^h(x)\}$ is shown in the next column. After performing the subtraction, we select the min value in the resulting matrix, and the result is placed in the dark blue matrix in the next column. Notice that the location where the result is placed corresponds to where the SE is on top of the input matrix. Lastly, before we slide the SE, we need to record the location of the elements in SE that produced the min value. Position matrix that has the same size as the SE is used for this purpose. All the elements of the position matrix are zeros except for one. This is shown in the last column of Figure 2.8. This position matrix is needed for the update of SE during the back-propagation. For each SE position, we have one position matrix. Therefore, the number of position matrices is equal to the number of

elements in the resulting matrix. The first row shows the position for the SE. The second row is when the SE slides one pixel horizontally and the third row is when the SE slides two pixels horizontally.

The same process is repeated for the miss operation, except that the min operation is replaced with the max operation. The details of the process are shown in Figure 2.9.

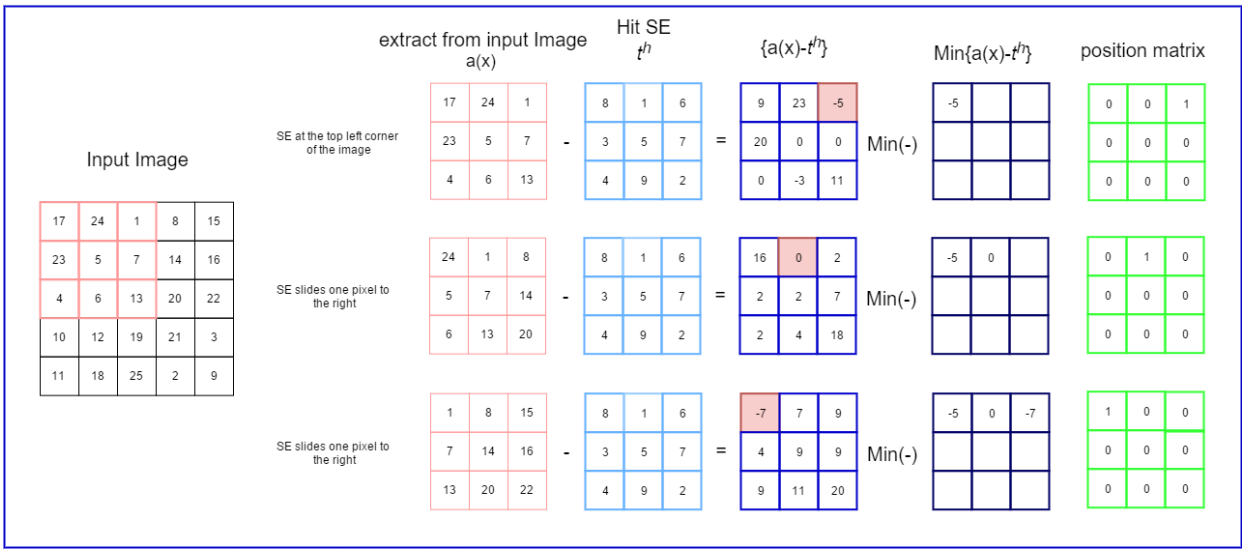


Figure 2.9: Grayscale hit operation

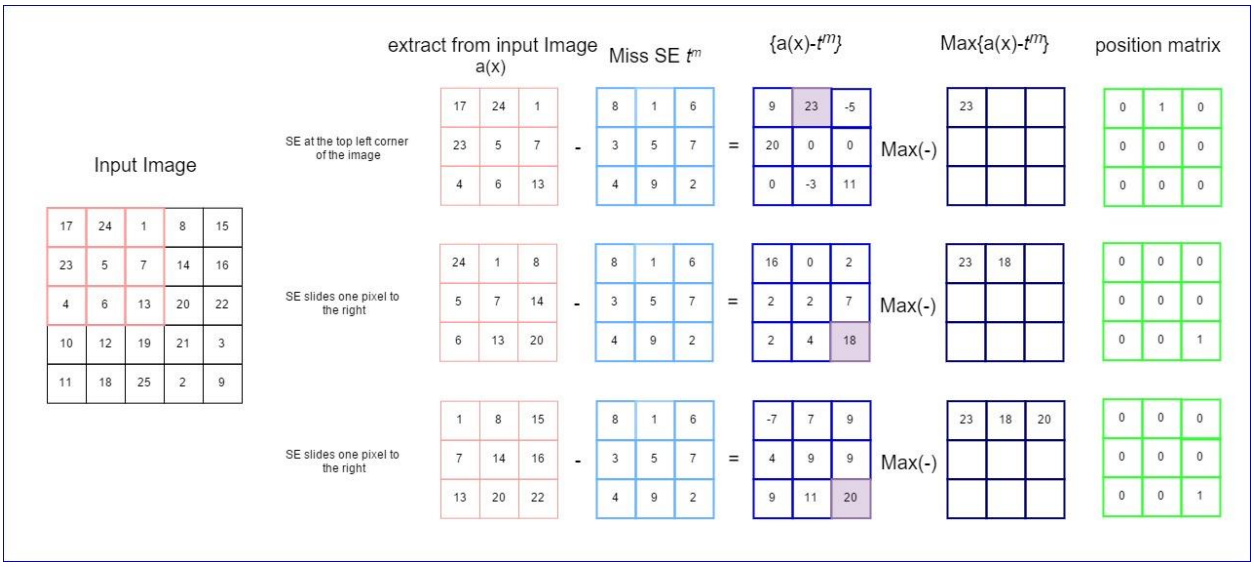


Figure 2.10 Grayscale miss operation

The hit-miss transform is performed on the output of the hit and miss operations. This output represents the feature map. A sample of hit-miss transform is shown in Figure 2.11.

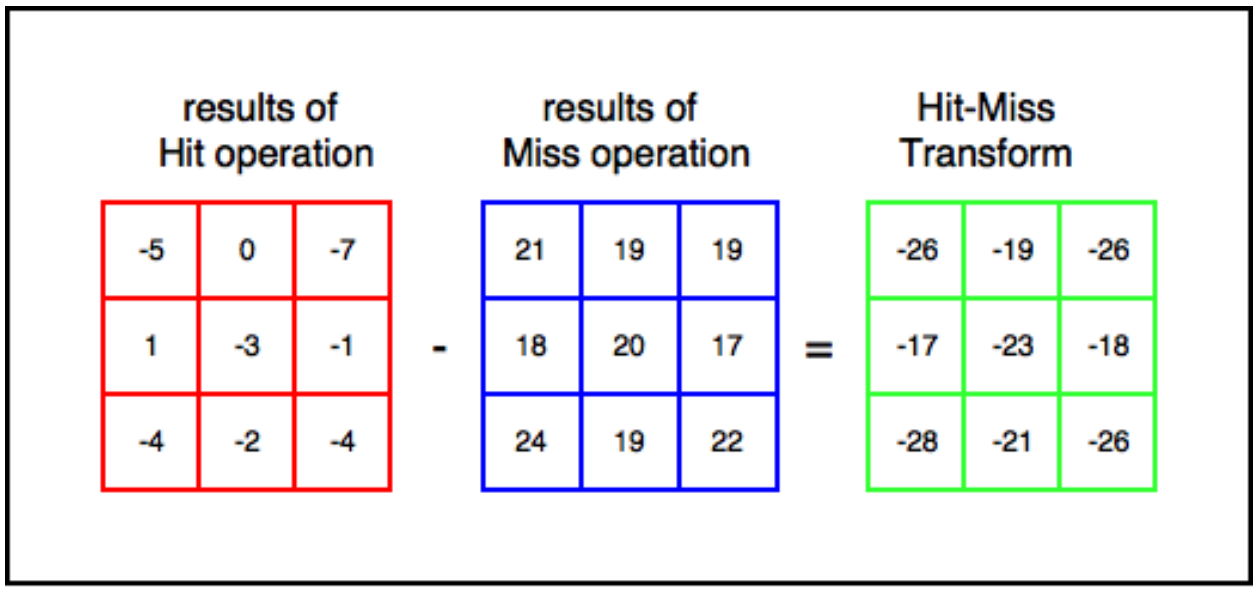


Figure 2.11: Grayscale hit-miss transform. Red matrix is the result of the Hit operation. Blue matrix is the result of the miss operation. Green matrix is the result of the hit-miss transform

Back-propagation:

The gradient descent optimization method is used for the MSNN, However, when using the chain rule for the derivation, we face a problem with our feature extraction layer. The min and max used in their respective equations are not differentiable. Won [4] offered a solution for this problem in his dissertation. The specificity of the hit and miss operations requires that the equations are derived in a non-traditional way. According to Won [4]:

The results of the derivation and their details can be found in.

$$\frac{\partial net_y^h}{\partial t_y^h(x)} = \frac{\partial}{\partial t_y^h(x)}\left[\underbrace{min}_{q\in D[t_y]}\{a(q) - t_y^h(q)\}\right]$$

$$= \begin{cases} -1 \; if \; x = \underbrace{\arg min}_{q=D[t_y]} \{a(q) - t_y^h(q)\} \\ 0 \qquad\qquad otherwise \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.30)$$

$$\frac{\partial net_y^m}{\partial t_y^m(x)} = \frac{\partial}{\partial t_y^m(x)} \left[\underbrace{max}_{q \in D[t_y]} \{a(q) - t_y^m(q)\} \right]$$

$$= \begin{cases} -1 \; if \; x = \underbrace{\arg max}_{q=D[t_y]} \{a(q) - t_y^m(q)\} \\ 0 \qquad\qquad otherwise \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.31)$$

$$\frac{\partial net_y^h}{\partial a(y)} = \frac{\partial}{\partial a(y)} \left[\underbrace{min}_{q \in D[t_k]} \{a(q) - t_k^h(q)\} \right]$$

$$= \begin{cases} 1 \; if \; y = \underbrace{\arg min}_{q=D[t_k]} \{a(q) - t_k^h(q)\} \\ 0 \qquad\qquad otherwise \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.32)$$

$$\frac{\partial net_y^m}{\partial a(y)} = \frac{\partial}{\partial a(y)} \left[\underbrace{max}_{q \in D[t_k]} \{a(q) - t_k^m(q)\} \right]$$

$$= \begin{cases} 1 \; if \; y = \underbrace{\arg max}_{q=D[t_k]} \{a(q) - t_k^m(q)\} \\ 0 \qquad\qquad otherwise \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.33)$$

The learning rule for the hit and miss are shown in Eq. (2.34) and (2.35), respectively.

$$\Delta \, t_y^h(x) = \eta \delta_y \frac{\partial net_y^h}{\partial t_y^h(x)} \ldots\ldots\ldots\ldots (2.34)$$

$$\Delta \, t_y^m(x) = -\eta \delta_y \frac{\partial net_y^m}{\partial t_y^m(x)} \ldots\ldots\ldots\ldots\ldots (2.35)$$

where η is the learning rate, and δ_y is the error delta.

For an MSNN with one feature extraction layer, δ_y is computed using Eq. (2.36)

$$\delta_y = \delta(y) = \sum_k \delta_k \, w_k(y) \quad \text{..... (2.36)}$$

w_k is the set of weights from the fully connected layer.

For an MSNN with more than one feature extraction layer, Eq. (2.36) is used for the top feature extraction layer. For the other feature extraction layers, we used Eq. (2.37).

$$\delta_y = \delta(y) = \sum_k \delta_k \left(\frac{\partial net_k^h}{\partial a(y)} - \frac{\partial net_k^m}{\partial a(y)}\right) \quad \text{..... (2.37)}$$

At first, these equations are confusing and intimidating. However, after examining them along with the equations of the forward pass, we can get an intuition on how to implement the back-propagation of the feature extraction layer.

Let's say we have an input image *I* of size 40 x 40 and a structuring element (SE) of size 5 x 5. For the forward pass, the SE slides over the input image from right to left and from top to bottom. At each pixel position, the hit and miss operations are performed as shown in Figures 2.8 and 2.9 respectively.

MSNN only updates the element of SE involved in the generation of min (max) at the pixel position. The hit element update depends on $\frac{\partial net_y^h}{\partial t_y^h(x)}$ as shown in Eq. (2.34). In Eq. (2.30) we can see that $\frac{\partial net_y^h}{\partial t_y^h(x)}$ is equal to -1 only for the element that generated the min. Otherwise, it is equal to 0. If more than one pixel is involved, only one pixel is updated.

The same process is followed for the update of the miss SE. According to Eq. (2.35), updating the Miss SE depends on $\frac{\partial net_y^m}{\partial t_y^m(x)}$. This term is equal to -1 for the element

that generated max only. Otherwise, it is equal to 0. If more than one pixel is involved, only one is updated.

The location of the element to update in both hit and miss is determined with the position matrix recorded during the forward pass (see Figures 2.9 and 2.10).

Figure 2.12 shows how this process is performed for a hit SE. The same thing applies to a miss SE. To facilitate the task, the back-propagated error is reshaped in the form of a matrix with the same dimensions as the feature map. This is shown as a red matrix. For each position of the SE during a forward pass, we have an error. For each error, we have a position matrix shown in green. The error is multiplied by the learning rate and the position matrix is then added to the hit SE.

The same process is followed for the miss SE, except that the update is subtracted instead of added as shown in Eq. 2.35.

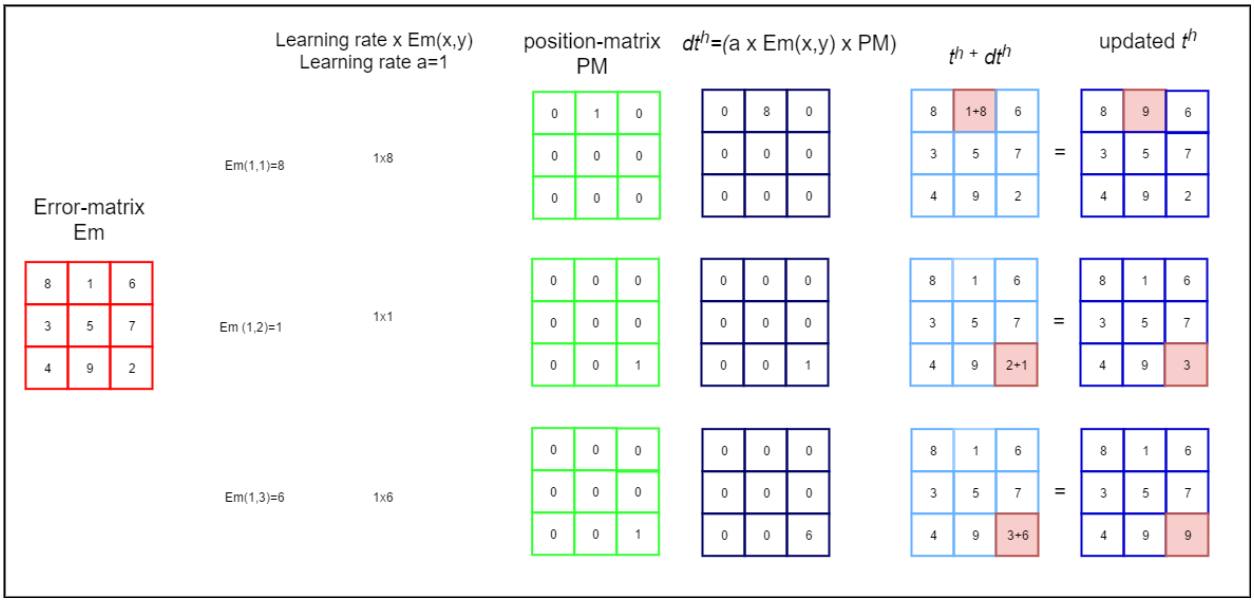


Figure 2.12 Update of the hit SE. The red matrix is the error matrix. The error is first multiplied by the learning rate. Green is the position matrix. The square highlighted in red is the element involved in the min. Therefore, it is the element to be updated.

D) Pseudocode for the MSNN

The pseudo code for the MSNN was presented in [19] and reproduced in Figure 2.13. The MSNN is executed in two phases. Phase one is the generation of sub-images, both positive and negative. Phase two is the training phase, including the forward pass and back-propagation through both the feature extraction layer, the max-pooling layer, and the neural network.

Phase 1:

1. The training dataset images are all read.
2. For each image, we define a window U such that the center of the target is inside this window.
3. Randomly generate positive sub-images where the center of the sub-image is always inside the window U.
4. Define a window V larger than the target such that the centers of the negative sub-images are randomly selected from outside the window V. This window is used to make sure that no negative sub-image has a part of the target in it.
5. The number of positive sub-windows and negative sub-windows are equal for each image. The newly obtained sub-images will be used to train our algorithm.

Phase 2:

The MSNN is trained in this phase.

1. A sub-image is randomly selected and passed through the forward pass. An error is calculated—then back propagated through the neural network and the feature extraction layer to update the structuring elements.

2. In the feature extraction layer, each element in the back propagated error is used to update one element of the SE as shown in Figure 2.10.
3. Each epoch has a number of iterations equal to the total of positive and negative sub-images for each image in the training dataset.
4. At the end of each iteration, the error of the current sub-image is compared to a predefined threshold. If the error is below it, then the sub-image is considered well learned and is replaced with another randomly selected sub-image of the same type (positive is replaced with a positive, and negative is replaced with a negative).
5. The training stops if the total training error of all the images is below a certain threshold for a consecutive number of times or if the maximum number of epochs is reached.

```
MSNN
Read N input image scenes S_1, ... S_N along with the center positions of the targets
in the scene
Epoch =1; RandSelect =0;
WHILE (RandSelect < MaxRandSelect And Epoch < MaxEpoch) DO
    FOR n=1 to N,
        Randomly select M target subimages with center inside U, T_n1, ... T_nM;
        Randomly select M background subimages with center outside V,
B_n1, ... B_nM;
        RandSelect =RandSelect+1;
    ENDFOR
    ErrMonitor =0;
    WHILE (Epoch < MaxEpoch AND ErrMonitor < ContinueLow) DO
        FOR subimage in {T_n1, B_n1, ... , T_nM, B_nM, }
            Perform forward and backward propagation;
            IF (PSS < WellTrained)
                Replace current subimage with new randomly selected one;
            ENDIF
        ENDFOR
        Epoch=Epoch+1;
        IF (RMSE < StopErr)
            ErrMonitor = ErrMonitor+1;
        ELSE
            ErrMonitor =0;
        ENDIF
    ENDWHILE
ENDWHILE
```

Figure 2.13: Pseudo code for the MSNN as shown in [19]

2.2.4 Literature review

Won's dissertation [4] is the basis for this work. It provides the theoretical background. In [4], the MSNN was also tested on vehicle detection under different occlusion conditions and found to be very effective. The pseudocode for the MSNN was provided by Won et al. in [19]. The MSNN was also employed in an ATR system. It provided TAP point (target-aim-point) and was tested on tank detection. These two documents are the main references for any work that involves the use of MSNN. Haun et al. [18] tested the MSNN under different conditions using four different architectures with the intention of improving object recognition versus processing time. It was found that the

more complex the architecture, the better the performance, but the processing time is longer which affects its ability to be used in an online application. The MSNN is mostly used for detection and/or recognition applications. Khabou et al. [29] used the MSNN for detecting faces. The database that Khabou created for this system has 10 images for each face (person). Comparison is done by calculating the average absolute error between the images in the database and the detected face. The face producing the lowest error is saved. Two thresholds are empirically defined, T1 and T2 where T1 < T2. If the average absolute error is less than T1 then we have a positive recognition. If the average absolute error is greater than T1 but less than T2 then the image is still considered a face, but it's classified as "unknown," whereas if it's greater than T2 then it's classified as a "false alarm". Sahoolizadeh et al. [30] explored the limitations of the MSNN performance for face recognition while altering one parameter at a time. It is worth noting that the structuring element in the morphological layer was not updated using the equations developed by Won. Instead, the MSNN was tested using "disk" and "diamond" structuring elements (SE). The performance of the MSNN was monitored using different sizes of the SE where it was found that it breaks if the size of structuring element is close or equal to the size of the target. The MSNN was also tested while varying the number of neurons in the hidden layer and determining the value of the learning rate. These values were found to be best determined empirically. Lala et al. [32] used the MSNN for face and eye detection and recognition of passport size images. Podder et al. [31] used SMQT (successive mean quantization transform) features and SNoW (sparse network of Windows) classifier to detect the faces. MSNN is used for this recognition process. However, flat SE elements are also used, i.e., the error is only back propagated through the neural network layers.

Chandrappa [33] used the Gabor filter feature extraction and a neural network to detect the faces. The candidate face is then fed into the MSNN for recognition. Similar to the MSNN described above, the back-propagation and update of weights is restricted to the neural network part of the MSNN.

Besides being used for face detection and recognition, the MSNN is also used in other types of detection. Cheng [34] used an MSNN to detect white blood cells. Cheng chose to fuzzify the MSNN. Since the white blood cells don't have a uniform shape, he modified the binary "hit" and miss operations by replacing "min" and "max" operations with equivalent fuzzy operations. Eq. (2.38) shows fuzzy hit and Eq. (2.39) shows fuzzy miss.

$$A \ominus B = \inf S(c(B(x-y)), A(x)) \quad \text{(2.38)}$$

$$A \oplus B = \sup T(B(x-y), A(x)) \quad \text{(2.39)}$$

T is a triangle norm operator such as t-norm; it replaces the max and *S* is the t-conorm and it replaces the min.

The pixel intensity in a binary image can be 0 or 1; whereas, the pixel intensity in a grayscale image can have any value in the interval [0.1]. Therefore, a grayscale image is treated as a fuzzy set.

Satellite images also attracted the use of MSNN with its ability to identify structures. Zheng [35] proposed a system based on the MSNN. It first starts with a preprocessing phase that includes the use of top-hat [64] and bottom-hat [64] techniques to eliminate noise and unwanted details from the image. Both these techniques are based on morphological operations (opening and closing).

$$Top - hat: TH(f) = f - (f \circ g) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.40)$$

$$Bottom - hat: BH(f) = (f \cdot g) - f \ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.41)$$

where, f is the image, g is the structuring element, $(f \circ g)$ is the opening operation, and $(f \cdot g)$ is the closing operation.

The next step is to convert the image to binary, so the Otsu algorithm is used to pick the most suitable threshold. The MSNN is then trained using sub-images extracted from the grayscale images. Both feature extraction layer and feedforward neural network layers are trained and updated using back-propagation.

2.3 Aerial and Satellite Image Applications

Aerial and satellite images have recently become the subject of many applications, especially since this data has been recently made available to the general public. These images may look very similar, but there are some differences between them. Among these applications, the one that gained popularity is vehicle detection. Different algorithms were developed to tackle this problem.

Pawar et al. [20] compared the performance of top-hat and bottom-hat techniques in detecting vehicles in highways. Zheng [21], [22] used top-hat and bottom-hat techniques to preprocess the images. Then, grayscale images were converted into binary images. The threshold is determined using the OTSU algorithm. The resulting images are fed into an MSNN. Cao et al. [23] obtained vehicle detectors trained with a super resolution algorithm using transfer learning. The detectors were originally trained on aerial images. Cao et al. used a linear SVM to tune them for satellite images. Zhou et al. [24] performed a series of preprocessing operations to isolate the area likely to have vehicles. Histogram equalization

and linear enhancement were used to obtain the best segmentation. Vegetation and water areas were suppressed using NDVI (normalized difference vegetation index) and NDWI (normalized difference water index), respectively. Combining these operations, the authors were able to extract the road where the vehicles were likely to be. Geometric characteristics were used to ensure that the road was extracted without any missing pieces. The vehicles were divided into two categories, bright and dark, so they were detected separately. For the bright vehicles, the OTSU algorithm was used to find the best threshold for the images. Then, the bright areas were detected as vehicles. For the dark vehicles, the result obtained from thresholding were inversed making it easier to detect these dark vehicles. Since the vehicles have varying brightness, the ones already detected in the first step were removed to prevent double detection of the same vehicle.

Wei et al. [25] focused on detecting cars in parking lots. The images were converted into edge maps using the Canny edge detector; then, they were preprocessed using erosion and dilation. Th vehicles were determined to be in 12 different orientations; therefore, 12 templates representing all the different positions were passed over each image and a correlation coefficient was computed. If a value higher than a threshold was obtained, then that candidate's sub-image was considered as a positive detection of a vehicle. Shu et al. [26] defined three categories of regions where vehicles can be found, highways, inner city roads and parking lots. Each category requires different parameters in order to detect the vehicles. The first step is to segment the images to extract the different regions. Each region is segmented using multi-scale resolution in order to transform the images into image objects. An initial classification of these image objects is done using an SVM. Spatial rules are used to optimize the final results. Abraham et al. [27] extracted 11 rules for a fuzzy

inference system from the grayscale images. Bright vehicles and dark vehicles were detected separately. Multi thresholding was used for detecting bright vehicles. OTSU thresholding was used for detecting dark vehicles.

CHAPTER 3. METHODOLOGY

3.1. Alternative MSNNs:

In an attempt to improve the performance of the MSNN, some modifications were developed to improve the performance of the MSNN. After examining the pseudocode in Section 2.2.3, we were able to see these modifications happening in Phase 2, Step 2 and in Phase 2, Step 4. In Phase 2, Step 2, we updated the SEs in the feature extraction layer. In Phase 2, Step 4, we replaced sub-images that were learned. A sub-image is considered well learned if it generates an error that is below a certain threshold.

We named the original algorithm "MSNN1," It can update one element for every element in the back-propagated error vector as explained in Figure 2.12. As for replacing a well learned sub-image, MSNN1 selects the new sub-image from a random set of sub-images in the same category. We also developed two alternatives, MSNN2 and MSNN3 by varying these two steps.

3.1.1 MSNN2:

In MSNN2, the modification occurs in Phase 2, Step 2, i.e., when updating the SEs. Instead of updating one element in SE per one element in the back-propagated error vector as shown in Figure 2.12. We recorded the location of all the elements of equal values in the forward pass, as shown in Figure 3.1. Then we updated them all during the back-propagation. When dealing with more than one element of equal value, which are either min or max, information can be lost if only one element is updated. This modification considers all the elements, which leads to improved target learning. This is illustrated in Figure 3.2. The forward pass and the back-propagation for MSNN2 is similar to MSNN1. The change happens when recording the location of an element generating min (max) as shown by the highlighted square in Figure 3.2.

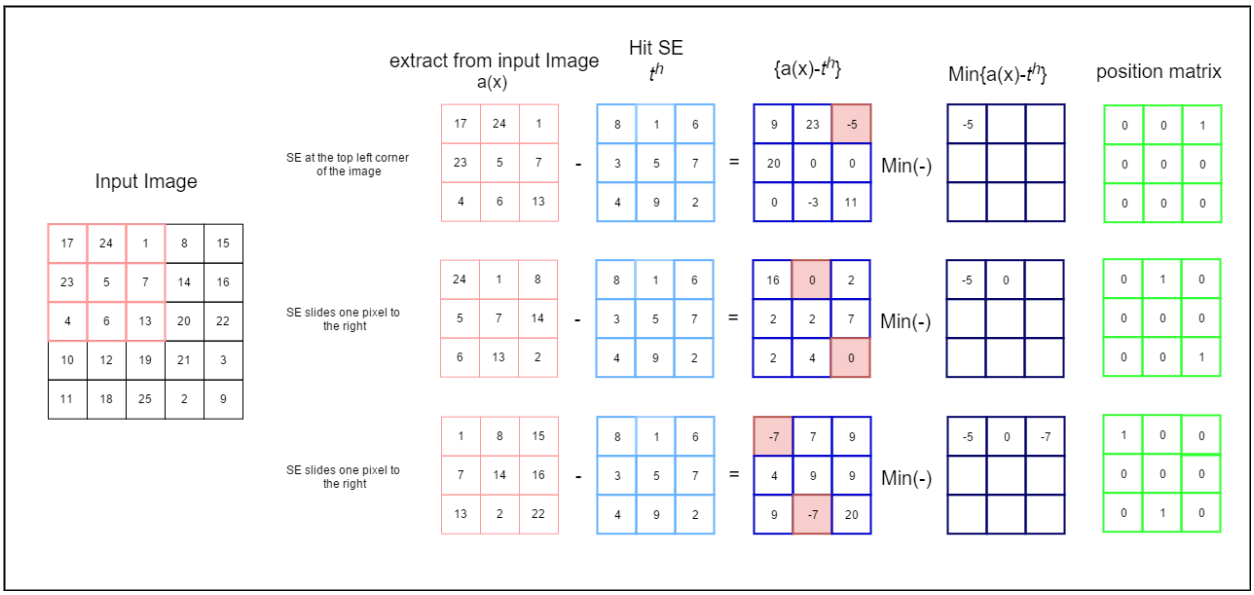


Figure 3.1: MSNN2 feature extraction layer forward pass, hit operation. Input image is processed with a 3x3 SE (red matrix). Highlighted squares in the blue matrix indicate the location of elements generating min value. Position matrix (green) records all the locations of these elements

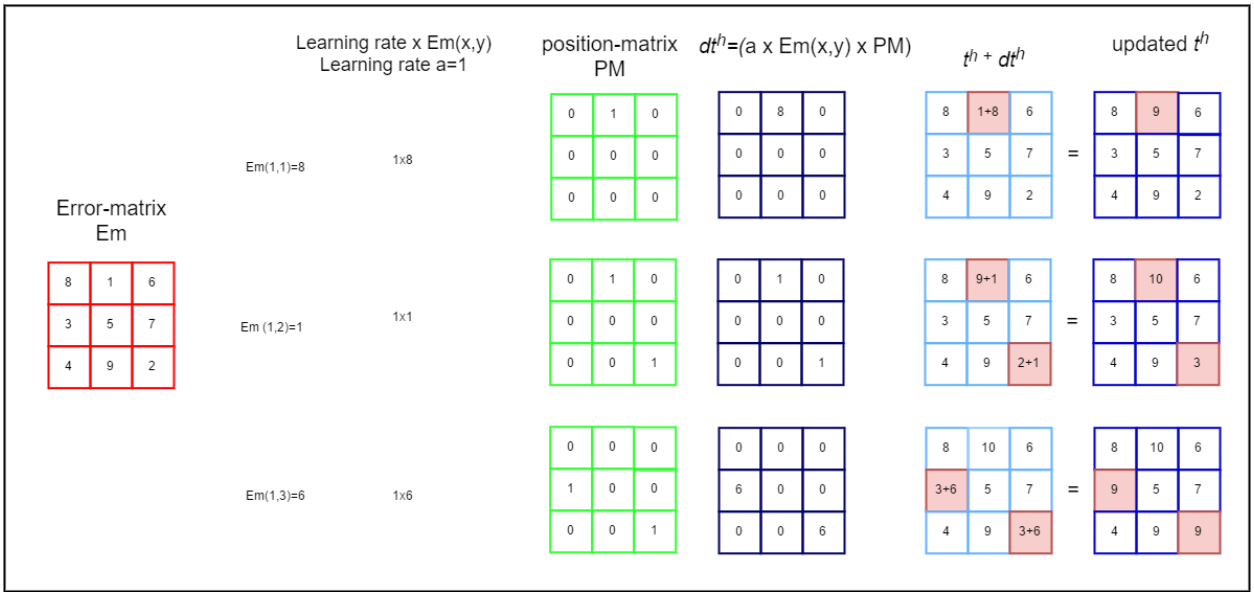


Figure 3.2: MSNN2 feature extraction layer structuring element (hit) update. Error value from error matrix (red) is multiplied by the learning rate (which equals 1 for simplicity). Position matrix (green) is used to make sure only corresponding elements are updated. Highlighted squares show the elements updated.

3.1.2 MSNN3:

For MSNN3, the modification occurs in Step 4 of Phase 2. This algorithm can still update one element in the SE just like MSNN1. However, when replacing the sub-image, we ran a test on the set of the possible sub-images, and we picked the one with the highest

error. This is more of a way to guide the MSNN to learn as much as possible about the target and background. This version of the MSNN is the slowest among the three.

The three elements will be tested under the same conditions to see which one better suits to be used for MIL framework.

3.2 Multiple Instance Learning

We had to adapt the MSNN algorithm to fit into the MIL framework. This task represented two challenges: 1) to develop a method allowing us to generate the bags and 2) to modify our algorithm with the capability to use bag labels instead of instance labels.

Since we were working with images, we considered each image as a bag; hence, we created one label per image. An image that contains the target is considered a positive bag and it is given the label '1'. An image that does not contain the target is considered a negative bag and it is given the label '0'. We used instances that were sub-images of the same size as our target. The MIL framework assumes the exact location of the target in the image to be unknown. Therefore, to make sure that at least one instance has the target in it, we used the "sliding window" technique. Thus, we were able to slide a window that has the same size as the target, and at each position we recorded an instance. If we were to do this pixel by pixel, a large bag would be generated, and the training time would be too long. Instead, we made the window jump a fixed number of pixels in a way that generated a reasonable number of instances per bag, and at the same time, included the whole target in at least one instance. This process is illustrated in Figure 3.3. This is obviously not an issue for a negative bag, but we wanted to make sure we learned as much as possible about the background.

The next step was to modify the MSNN algorithm. We called the new algorithm "MI-MSNN" which stands for Multiple Instance MSNN. Since a label is assigned for each bag, we had to pass the data as bags. This means that all the instances of a bag were evaluated in the forward pass. For the back-propagation phase, a number of approaches were proposed. Since we were 100% confident that all the instances in a negative bag would be negative, only one straight forward approach was required for the negative bag. The challenge lay in the positive bag.

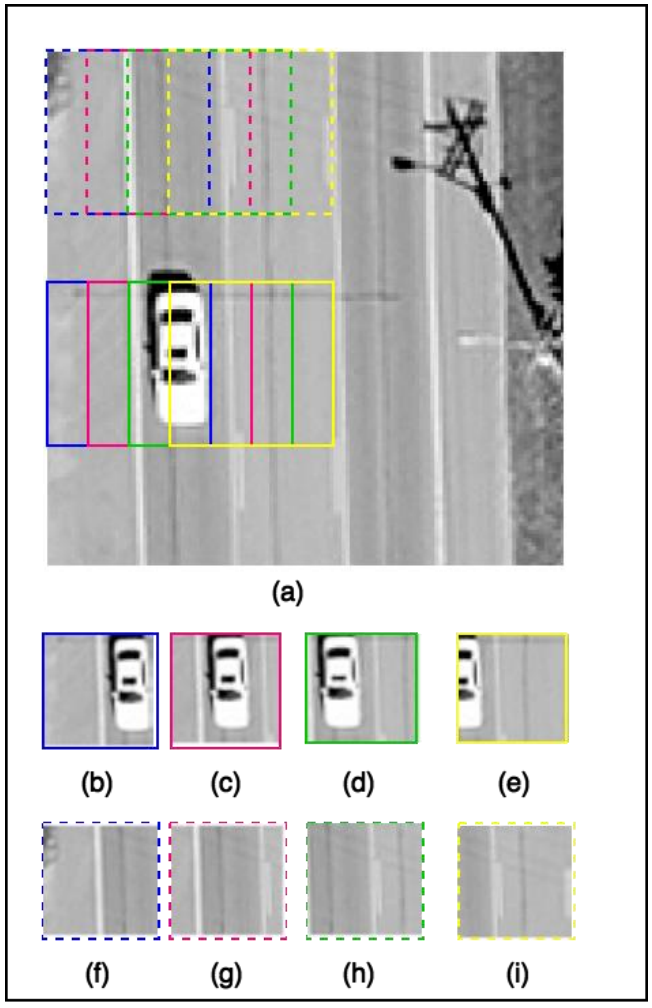


Figure 3.3: Generating Instances from an Image using the Sliding Window Technique. (b)- (e) are positive instances recorded each time the window slides a fixed number of instances. (f)- (i) are negative instances recorded each time the window slides a fixed number of instances. Together, these instances form a positive bag

For the first approach, we knew the label of the bag is the label of each of its instances. Therefore, we back propagated an error for each instance. However, there were a few issues with this approach. First, it is computationally expensive to do this for each bag, considering the way the bags are generated. Each bag has hundreds of instances, and evaluating them all in the forward pass as well as in the back-propagation makes the training too long. This problem was solved by effectively implementing the algorithm in parallel form. Assigning the bag label to all its instances was not a problem for a negative bag since all of them were negative. A positive bag had target and background instances that could not be distinguished precisely. So, assigning an all-inclusive bag label to all of them could only confuse the algorithm.

Based on the definition of the positive bag, to classify a bag as positive, we need one instance to generate confidence as close to '1' as possible. So, the first approach is to look for the maximum instance in each bag and try to maximize it as much as possible. This approach is called "MI-SM-MSSN" where SM stands for select maximum. The pseudocode for this approach is shown in Figure 3.4 The issue with this approach is that this algorithm may get stuck on one instance which may or may not be the actual target. Our second approach works around this problem. Let's say we have 'N' instances in our positive bag. Instead of selecting the instance with maximum confidence, we select the top 'n' instances with the highest confidence, where $n < N$; then, we randomly pick one of them. This approach gives the algorithm a chance to avoid getting stuck with the wrong instance early on. This approach is called "MI-RSM-MSNN" where RSM stands for roulette select maximum. The pseudocode is shown in Figure 3.5

Since which instance is selected in a negative bag is not of concern here, we used the roulette select maximum (RSM) approach when selecting instances for the negative bags in both methods.

MI-SM-MSNN

Read the Bags: $B_1^+, B_2^+, \dots B_N^+, B_1^-, B_2^-, \dots B_M^-$
Read the labels: $L_1^+, L_2^+, \dots L_N^+, L_1^-, L_2^-, \dots L_M^-$
Initialize Structuring Elements (SEs)
Initialize Neural Network Weights
Initialize all other parameters:
    number of neurons in the hidden layer, activation function parameter, learning rate, StopE, …
Epoch=1;
While (Epoch< Epoch_max && RMSE>StopE)
For i= 1: N+M
    Randomly select a Bag B_r^* and its label L_r^*
Pass all instances through feature extraction layer (Hit-Miss)
Pass all instances through forward pass of the Neural Network and get Y_r
$Y_r = [y_1, y_2, \dots y_S]$ % S= number of instances in the selected Bag
    If $L_r^* = 0$
Sort Y_r 'ascending'
$temp = [y_{S-n+1}, \dots y_S]$
Randomly select y_i from *temp*
    Elseif $L_r^* = 1$
$\max(y_r)$
    end If
    compute Error *'e'*
    perform the back propagation
    update Neural Network Weights and SEs
    $E = E + e^2$
end For
        $RMSE = \sqrt{E}$
Epoch=Epoch+1
end While

Figure 3.4 Pseudocode for SM-MSNN

MI-RSM-MSNN

Read the Bags: $B_1^+, B_2^+, \ldots B_N^+, B_1^-, B_2^-, \ldots B_M^-$
Read the labels: $L_1^+, L_2^+, \ldots L_N^+, L_1^-, L_2^-, \ldots L_M^-$
Initialize Structuring Elements (SEs)
Initialize Neural Network Weights
Initialize all other parameters:
    number of neurons in the hidden layer, activation function parameter, learning rate, StopE, …
Epoch=1;
While (Epoch< Epoch_max && RMSE>StopE)
For i= 1: N+M
    Randomly select a Bag B_r^* and its label L_r^*
Pass all instances through feature extraction layer (Hit-Miss)
Pass all instances through forward pass of the Neural Network and get Y_r
$Y_r = [y_1, y_2, \ldots y_S]$ % S= number of instances in the selected Bag

Sort Y_r 'ascending'
$temp = [y_{S-n+1}, \ldots y_S]$
Randomly select y_i from *temp*
    compute Error *'e'*
    perform the back propagation
    update Neural Network Weights and SEs
    $E = E + e^2$
end For
        $RMSE = \sqrt{E}$
Epoch=Epoch+1
end While

Figure 3.5 Pseudocode for RSM-MSNN

CHAPTER 4. EXPERIMENTS AND RESULTS

4.1MSNN

The performance of the two alternative MSNNs are verified using a dataset of 27 aerial images as the training set. Each image contains one vehicle. A sample of these images is shown in Figure 4.1. The images are of size 128 x 128, and from each image there are 40 positive sub-images and 40 negative sub-images, all the sub-images are 40 x 40. The network parameters are the same for the three algorithms and are summarized in Table 4.1 below

Table 4.1 Network parameters for the three versions of the MSNN

Parameters	**MSNN1**	**MSNN2**	**MSSN3**
Number of SE	One pair of hit and miss	One pair of hit and miss	One pair of hit and miss
Size of SE	5x5	5x5	5x5
Number of neurons in hidden layer	40	40	40
Learning rate	$\begin{cases} 1 \le epoch < 1500 \dots\dots\dots\dots\dots\ eta = 0.5 \\ 1500 \le epoch < 4000 \dots\dots\dots\ eta = 0.2 \\ 4000 \le epoch < 6000 \dots\dots\dots\ eta = 0.1 \end{cases}$		
Activation function type in neural network	Sigmoid function	Sigmoid function	Sigmoid function
Parameters of the activation function	$\frac{1}{1+e^{-0.5X}}$	$\frac{1}{1+e^{-0.5X}}$	$\frac{1}{1+e^{-0.5X}}$
Size of sub-images			
Number of sub-images	40 positive/ 40 negative	40 positive/ 40 negative	40 positive/ 40 negative
Number of epochs	6000	6000	6000

We will use ROC curves to evaluate the performance of the three algorithms. The original algorithm (MSNN1) is compared to one of the two alternatives, one at a time. For each comparison, we look at the ROC curve. On each curve, we also plot the threshold producing the True Positive Rate (TPR) and the False Positive (FP). The color code displays the change in thresholds. We focus on the area below the two false alarms.

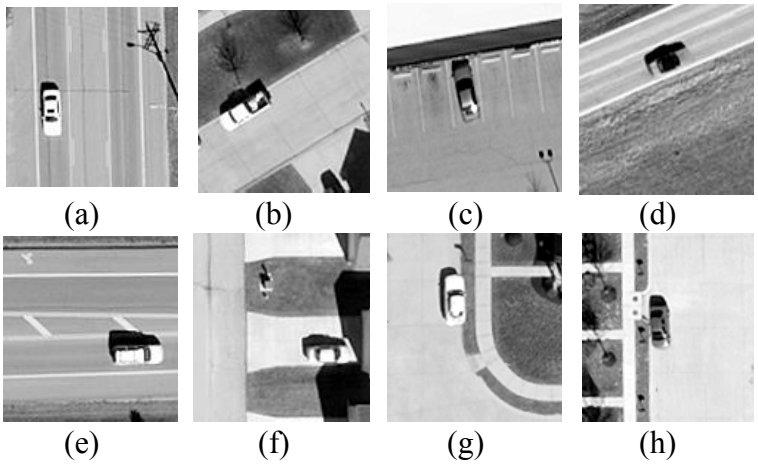


Figure 4.1: Sample of training images vehicles

The dataset used for the test is composed of 76 images from which 32 are positive and 44 are negative. In the positive images, 25 contain one vehicle, six contain two vehicles separated from each other, and one image contains three vehicles. Samples of the positive images are shown in Figure 4.2 while Figure 4.3 shows samples of negative images

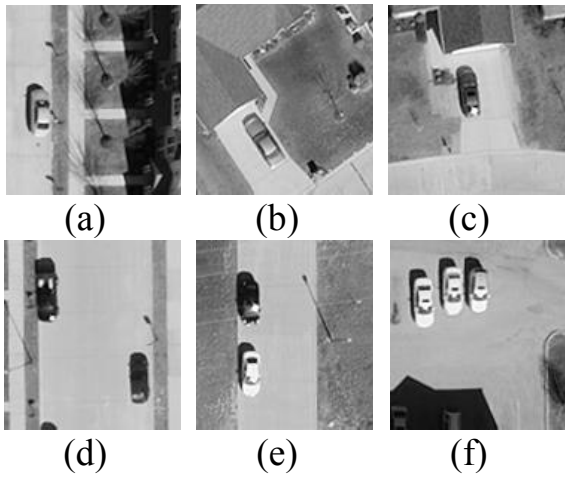


Figure4.2: Sample of positive test images from Dataset1. (a), (b), and (c) images have one vehicle each. (d) and (e) images have two vehicles sparsely arranged. (f) images have three vehicles

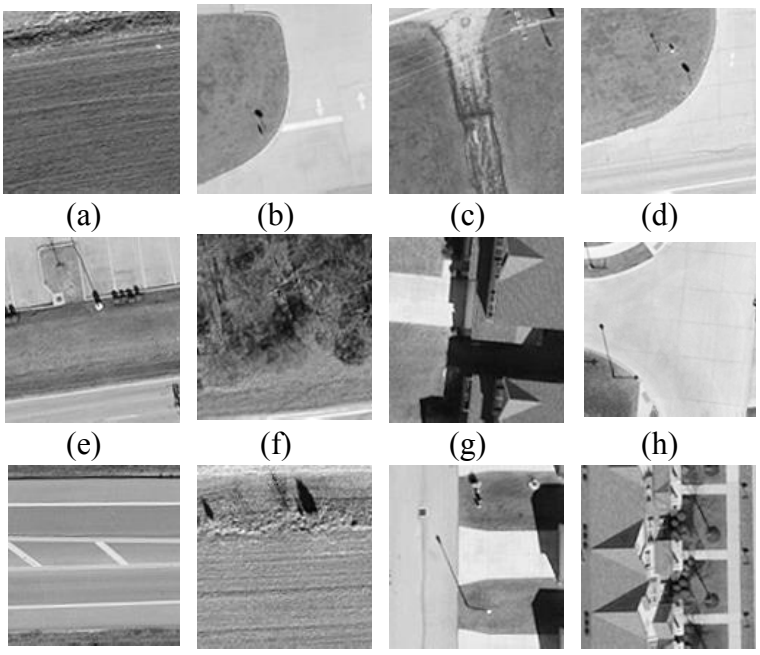


Figure4.3: Samples of negative test images from Dataset 1

MSNN1 vs MSNN2:

Figure 4.4 shows the ROC curves for MSNN1 (update one element at a time) in green, and MSNN2 (update all the possible elements each time) in blue. Figure 4.5 shows the ROC curves plotted for the FP range [0.2]. We see that MSNN1 reaches a 95% true positive rate (TPR) with one false alarm per image while MSNN2 has around four false alarms when it reaches 95%. However, a closer look at the curves in Figure 4.6 provides more information. Using the color bar as a reference we can see that the threshold that generates a 70% detection in MSNN1 and 0.6 false alarm (about three false alarms every five images) is the same threshold that generates 90% detection in MSNN2 with 1.8 false alarm (about nine false alarms every five images).

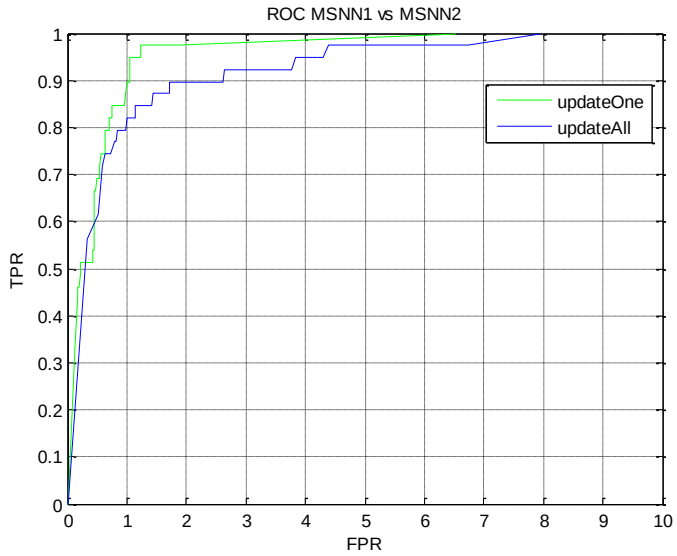


Figure 4.4: ROC curves for MSNN1 (green) and MSNN2 (blue)

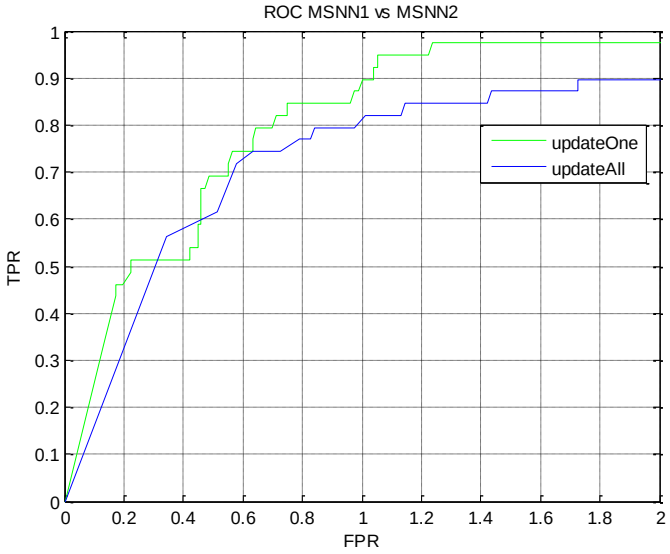


Figure 4.5: ROC curve for MSNN1 (green) and MSNN2 (blue). Plotted for FP= [0,2]

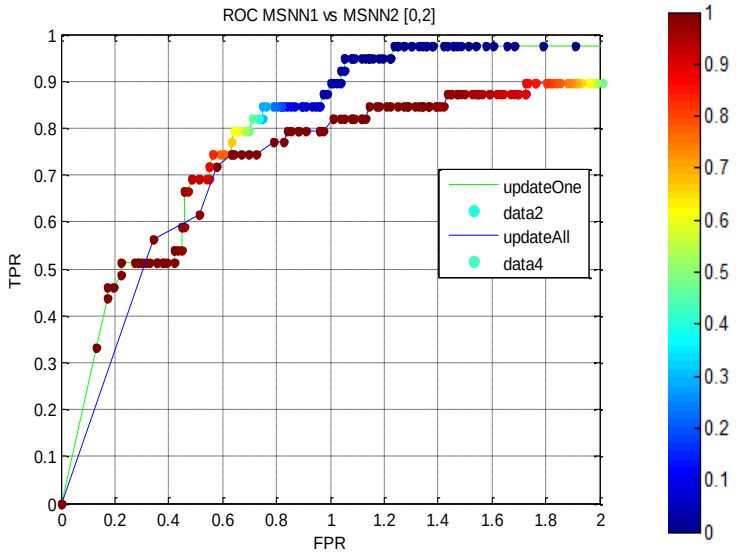


Figure 4.6 ROC curve for MSNN1 (green) and MSNN2 (blue). Plotted for FP= [0,2]. The colors on the curve indicate the threshold. The color bar to the right shows the corresponding value of the threshold.

MSNN2 achieves higher detection than MSNN1 (90% vs. 70%), but it also generates a higher number of false alarms.

Figure 4.7 shows several detection samples from MSNN1 while Figure 4.8 displays detection samples from MSNN2. These images are taken at a confidence (threshold) equal to 0.9. The center of the target is determined using a dot, and then a bounding box is drawn around the entire target with the dot as its center. The blue color signifies that this is the maximum detection in the image, whereas the green color signifies a detection higher than the threshold used (0.9 in this case). If no confidence is higher than the threshold, then no dots or bounding boxes are drawn. It is worth noting that non-maxima suppression (NMS) was used to eliminate multiple detections of the same target. Since our target is 40 x 40, any confidence within 16 pixels from our target was suppressed. The reason for using 16 pixels is related to the shape of the target. The vehicles have a rectangular shape, which means that if we used a diameter of more than 16 pixels, we risked losing some targets.

The first observation is that whenever a detection occurs, the maximum confidence is always on a vehicle, which proves that both algorithms are learning the target. The vehicles detected are in different orientations and have different colors. We also notice that MSNN1 missed the vehicles in Figure 4.7(b) and Figure 4.7(h). In Figure 4.7(b) the vehicle is white and oriented diagonally, while Figure 4.7(h) has two black vehicles oriented horizontally.

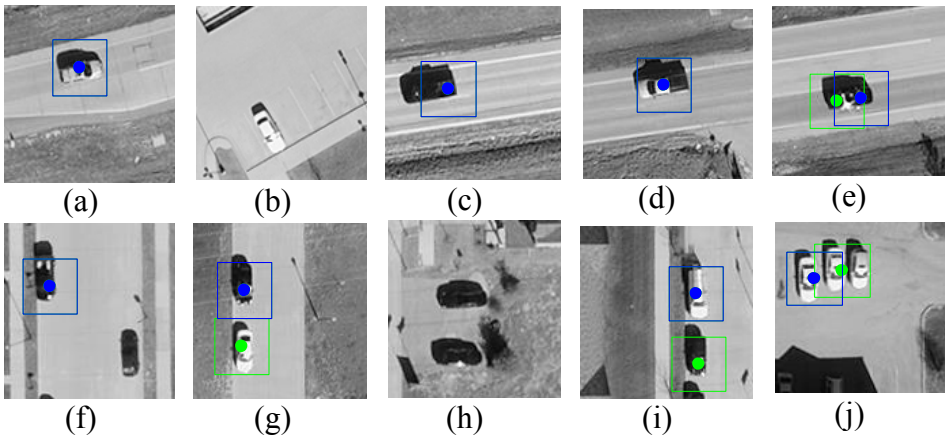


Figure4.7: MSNN1 detection samples for positive images. Vehicle are detected in (a), (c)–(g), (i), and (j). Vehicles are not detected in (b) and (h).

All vehicles in Figure 4.8 were detected with maximum confidence. False alarms were detected in Figures 4.8(g)–(i). These false alarms could have been eliminated using NMS if a diameter bigger than 16 was used.

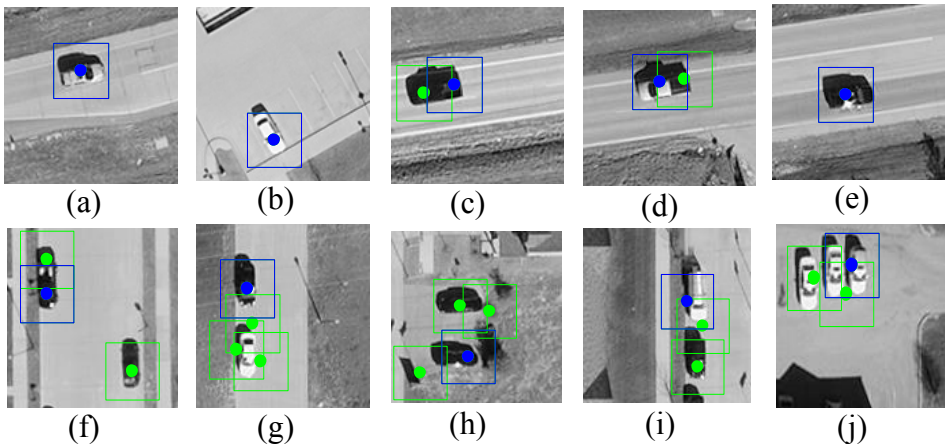


Figure 4.8: MSNN2 detection samples for positive images. Vehicles are detected in all images. False alarms were produced in (g), (h) and (i).

All targets are detected for the MSNN2, whereas not all targets are detected in MSNN1. On the other hand, MSNN2 produced more false alarms than MSNN1.

MSNN1 vs MSNN3:

Figure 4.9 shows the ROC curves for MSNN1 (updated one element at a time) in green, and MSNN3 (updated one element and replaced with highest false alarm) in red. It appears that MSNN1 dominates MSNN3 in both detection rate and number of false alarms per image. However, when we look at Figure 4.10, where we only focus on the FP range [0,2], we see a very close performance.

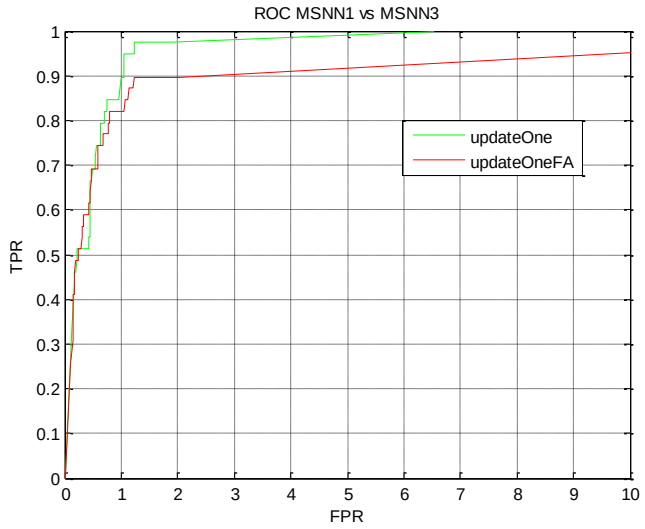


Figure 4.9: ROC curve for MSNN1 (green) and MSNN3 (red)

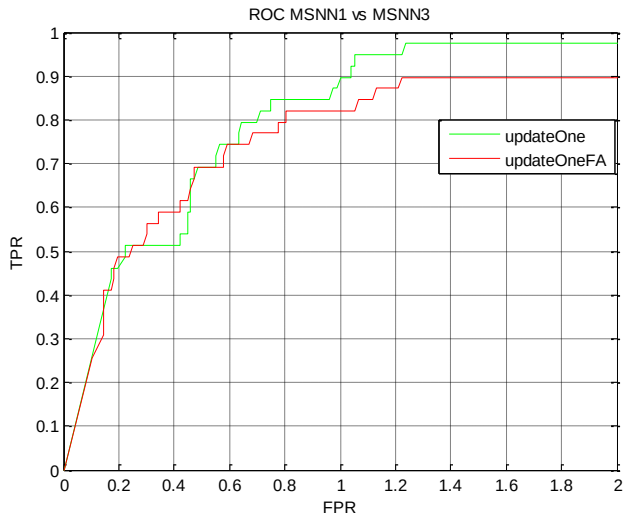


Figure 4.10: ROC curve for MSNN1 (green) and MSNN3 (red). Plotted for FP= [0,2]

Considering the threshold values in Figure 4.11, we see a close performance for the two algorithms in both detection rate and number of false alarms per image. At the threshold equal to 0.9, MSNN1 achieves 70% TPR while generating 0.6 false alarms (three false alarms every five images), whereas MSNN3 achieves 60% TPR while generating 0.4 false alarms (two false alarms every five images). MSNN1 wins in detection but loses in false alarm rate by a small margin

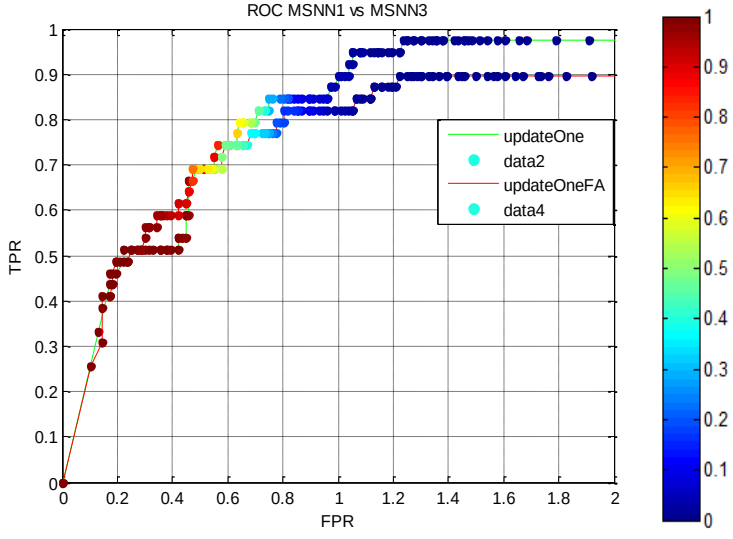


Figure 4.11: ROC curve for MSNN1 (green) and MSNN3 (red). Plotted for FP= [0,2]. The colors on the curve indicate the threshold. The color bar to the right shows the corresponding value of the threshold.

Figure 4.12 shows detection samples for MSNN3 for the same images of MSNN1 shown in Figure 4.7. First, we start by detection: MSNN1 and MSNN2 had identical performance in (a)–(g). MSNN1 detected all vehicles in (i) and (j), while MSNN3 missed one of the three vehicles in (j) and both vehicles in (i). On the other hand, MSNN3 detected one of the two vehicles in (h) while MSNN1 missed both. Both algorithms did not produce any false alarms in these images.

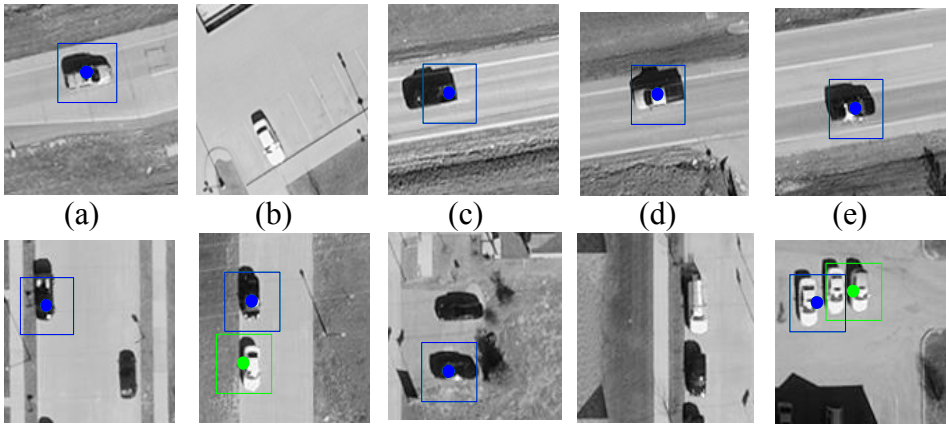


Figure 4.12 MSNN3 Detection samples for positive images. Vehicles are detected in (a), (c), (d), (e), (g). No vehicle is detected in (b), (i). (f), & (h); however, (j) has multiple vehicles, but not all were detected.

To summarize, MSNN1 and MSNN3 had close performances in both detection rate and number of false alarms per image. MSNN2 had a higher detection rate than MSNN1 and MSNN3, but it produced a higher false alarm rate per image compared to them.

Among the three algorithms, we believe that MSNN2 is the most suitable for our multiple instance learning problem. The reason for this decision is that with the weakly labeled data we have in our MIL framework, we need all the detection ability we can get, especially since our instances are just sub-images with no preprocessing or specification of target centers.

4.2 Comparing MSNN and CNN Algorithms in the MIL framework

Among computer vision algorithms, the convolutional neural network (CNN) is the closest in structure and optimization method to MSNN. As stated in Section 2.2.1, CNN is composed of three layers: the convolutional layer, the pooling layer, and the fully connected layer. CNN is similar to MSNN in the pooling layer and in the fully connected layer. However, it differs in the feature extraction layer. CNN uses the convolution operation to learn the features, whereas MSNN uses morphological operations to perform the same task. In these experiments, we compare the two algorithms to see which one has a better feature learning layer in the multiple instance learning framework.

SM-MSNN is compared with SM-CNN, while RSM-MSNN is compared with RSM-CNN. An aerial images dataset was used to train all four algorithms. To ensure a fair comparison, all the common parameters are identical across all four algorithms. These parameters are summarized in Table 4.2 below.

Table 4.2 Experiment 1 Network Parameters for MIL-MSNN and MIL-CNN

Parameters	SM-MSNN	SM-CNN	RSM-MSNN	RSM-CNN
Number of SE / kernels	One pair of hit and miss	One pair of hit and miss	2 kernels	2 kernels
Size of SE/ kernels	5x5	5x5	5x5	5x5
Activation function for feature extraction layer	Not applicable	Not applicable	Sigmoid function	Sigmoid function
Parameters for feature extraction activation function	Not applicable	Not applicable	$\frac{1}{1+e^{-0.5X}}$	$\frac{1}{1+e^{-0.5X}}$
Number of neurons in hidden layer	40	40	40	40
Learning rate	$\begin{cases} 1 \le epoch < 2500 \dots\dots\dots\dots\dots\ eta = 0.5 \\ 2500 \le epoch < 6600 \dots\dots\dots\ eta = 0.2 \\ 6600 \le epoch < 10000 \dots\dots\ eta = 0.1 \end{cases}$			
Activation function type in neural network	Sigmoid function	Sigmoid function	Sigmoid function	Sigmoid function
Parameters of the activation function	$\frac{1}{1+e^{-0.5X}}$	$\frac{1}{1+e^{-0.5X}}$	$\frac{1}{1+e^{-0.5X}}$	$\frac{1}{1+e^{-0.5X}}$
The size of instances	40 x 40	40 x 40	40 x 40	40 x 40
Number of bags positive/negative	34/43	34/43	34/43	34/43
Number of epochs	10000	10000	10000	10000

As stated in Section 3.2, the way the bags are created is straightforward. The "sliding window" approach is used as shown in Figure 3.3. Both positive and negative bags are generated in the same way.

A window the same size as the target (40 x 40) slides horizontally and vertically over the image. An instance is recorded every 10 pixels. Each image generates one bag only. We have 34 positive images and 43 negative images, so our training dataset is made of 34 positive bags and 43 negative bags. Since the images are 128 x 128, each bag contains 81 instances. The positive images are similar to the images shown in Figure 4.1. The negative images are similar to the images shown in Figure 4.3. Each of the four algorithms is trained

five times using a different random initialization. For each run, the initialization of the SE/kernels is the same for the four algorithms.

4.2.1 Bag Classification:

To evaluate the results, we test every image by generating instances in the same way that bag instances are created. However, during the test an instance is recorded every four pixels. The instance with the highest confidence is a representative for the entire bag. ROC curves are used to assess the performance of each algorithm in a bag classification. We assessed the overall performance of the four algorithms over the five runs. We also compared the average ROC curves for each algorithm. We selected two methods to generate an average ROC curve. The first method is called vertical averaging: For each value in the X-axis, an average of the five values in the Y-axis is computed; then, a curve is plotted. In addition to the average curve, an envelope representing the minimum and the maximum value at each point is also plotted. The second method is called the cumulative ROC curve. All the outputs obtained from the five runs are used to plot one ROC curve. The ROC curves obtained from both approaches are also compared.

When plotting the ROC curves, we have the true positive rate (TPR) on the Y-axis and false positive (FP) on the X-axis. This allows us to count the number of misclassified bags for each TPR value. When curves from the vertical averaging and cumulative ROC curve are plotted together, the X-axis is changed to a false positive rate (FPR) to make sure they are fairly compared.

4.2.1.1 SM-MSNN and SM-CNN

The obtained results were tested on the training data, and the results are shown in the figures below. For the SM case, we have the five runs and the vertical average curves

for SM-MSNN in Figure 4.13. The ROC curves for the 5 runs of the SM-CNN are shown in Figure 4.14, along with the vertical averaging curve.

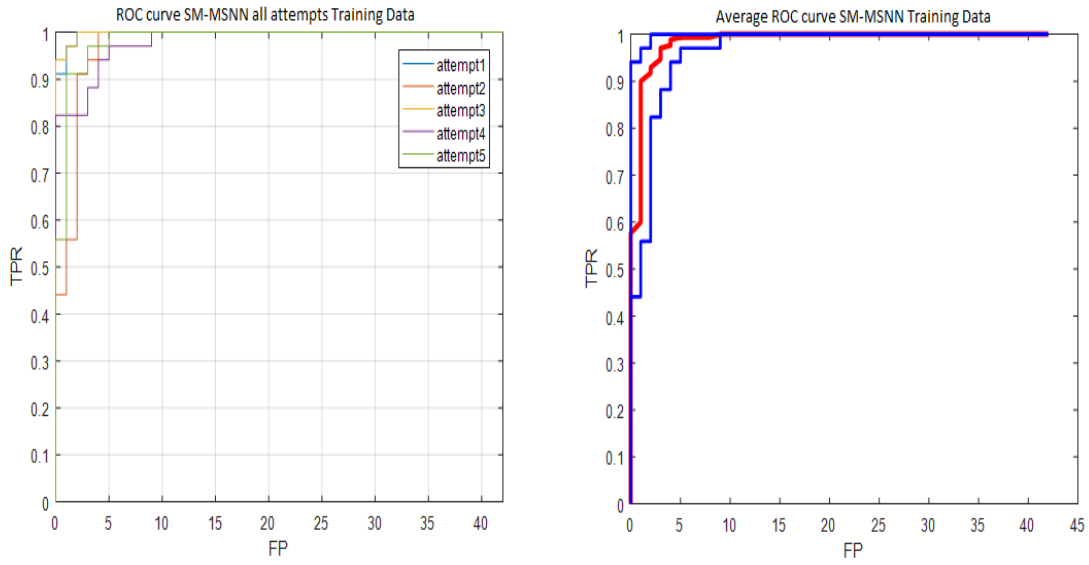


Figure 4.13 SM-MSNN ROC curves on training data. On the left, each color represents one of the 5 runs. On the right, the vertical averaging ROC curve is plotted in red and the envelope in blue.

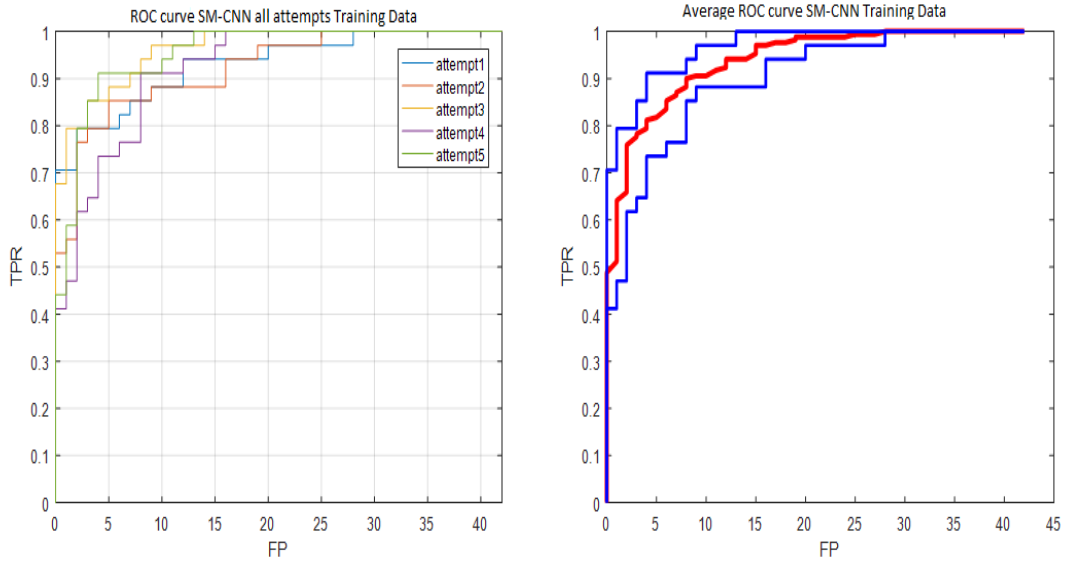


Figure 4.14 SM-CNN ROC curves on training data. On the left, each color represents one of the five runs. On the right, the vertical averaging ROC curve is plotted in red and the envelope in blue.

These results are plotted together in Figure 4.15 for comparison. The SM-MSNN curves are displayed in red and the SM-CNN curves are displayed in blue. We can see a

better performance for the SM-MSNN in most of the cases. This is again shown in Figure 4.16 with the vertical averaging curves for both algorithms, as well as the cumulative ROC curve. SM-MSNN reaches 100% detection with 5 FP bags whereas SM-CNN reaches 100% detection with 20 FP bags. A similar behavior is observed in the cumulative ROC curve. SM-MSNN is clearly better than the SM-CNN. Also, we can see that the results obtained from cumulative ROC curves are equal to the results obtained from the vertical averaging curve multiplied by 5. Keeping in mind that the cumulative ROC curve uses all the data from the five runs simultaneously, these results makes sense.

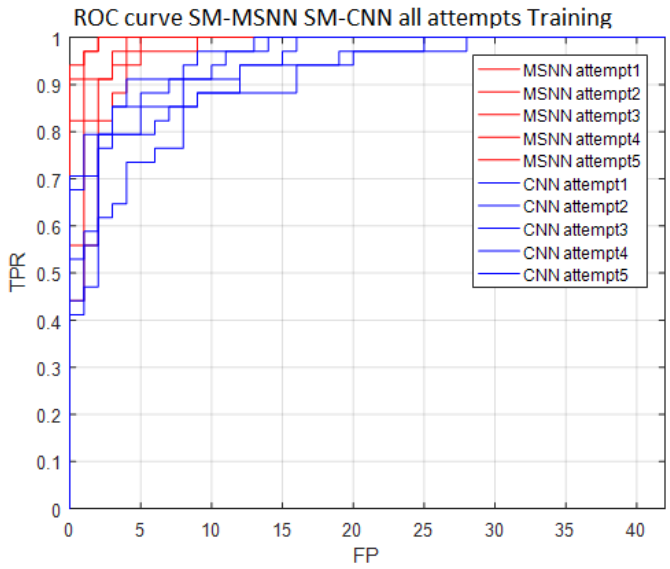


Figure 4.15 All ROC curves of the training data for the five runs. SM-MSNN (red) and SM-CNN (blue)

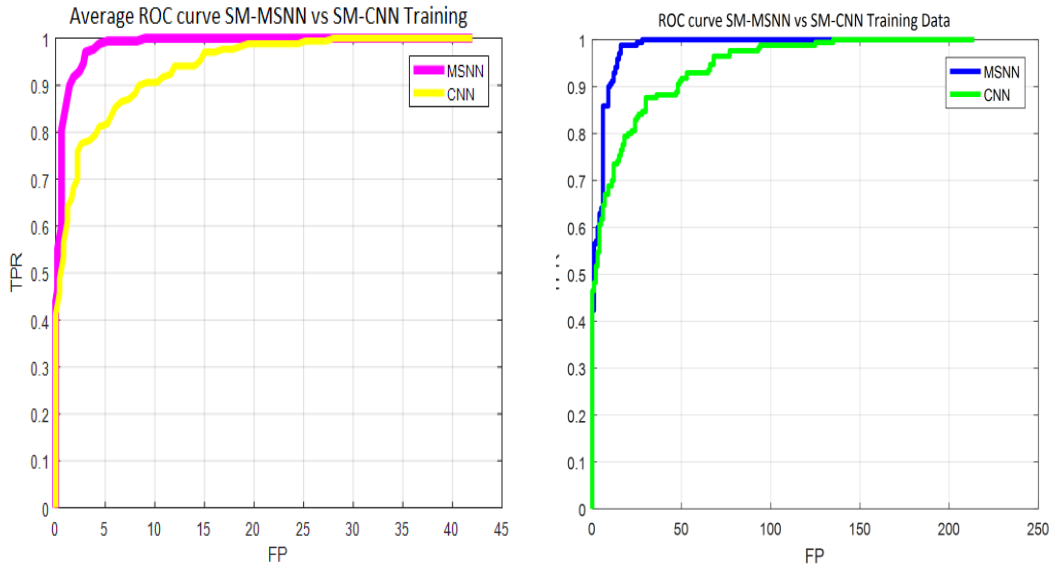


Figure 4.16 SM-MSNN and SM-CNN average ROC curve comparison on training data.
On the left, vertical averaging shows SM-MSNN in pink and SM-CNN in yellow.
On the right, cumulative ROC curve shows SM-MSNN in blue and SM-CNN in green.

Figure 4.17 shows the vertical averaging curves and the cumulative ROC curves for both SM-MSNN and SM-CNN. We changed the X-axis to a false positive rate (FPR) to ensure a fair comparison because the vertical averaging curve is plotted on the span of one run only whereas the cumulative ROC curve is plotted on the span of five runs. This makes the comparison unfair for both curves. For the SM-MSNN both the vertical averaging curve and cumulative ROC curve are nearly the same with some minor differences. The same can be said about the SM-CNN.

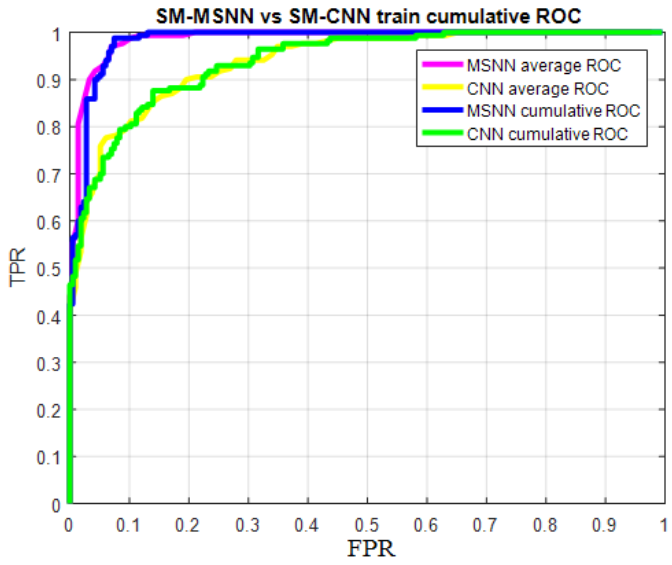


Figure 4.17 Results for training data show SM-MSNN vertical averaging curve as pink. SM-MSNN cumulative ROC curve is blue. SM-CNN vertical averaging curve is yellow. SM-CNN cumulative ROC curve is green

The first test data set is composed of 76 images (bags), where 32 are positive and 44 are negative. From the positive images, 25 have one vehicle, six have two vehicles arranged at a fair distance from each other, and one image has three vehicles. This is the same dataset used for testing the MSNN algorithms.

Figure 4.18 shows the ROC curves depicting the outcome of testing using the results from the five runs of SM-MSNN. The vertical averaging ROC curve is also shown in the same figure. Figure 4.19 shows the SM-CNN where ROC curves are depicting the testing results from the five runs compared with the vertical averaging ROC curve.

The test results from the five runs are shown together in Figure 4.20 for both algorithms. It is not clear which of the two algorithms is superior. The best ROC curve is for one of the SM- MSNN runs, but the worst case also belongs to an SM-MSNN run. By

looking at the average curve for each algorithm with its envelope, we see that the SM-CNN is more compact whereas MS-MSNN is more spread.

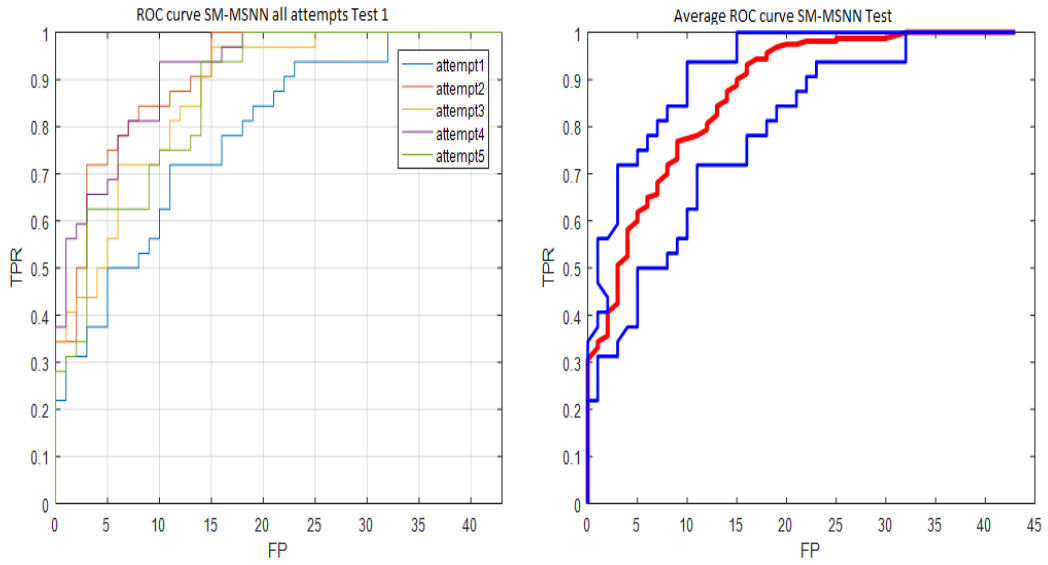


Figure 4.18 SM-MSNN ROC curves on test data1. On the left, each color represents one of the five runs. On the right the vertical averaging ROC curve is plotted in red and the envelope in blue

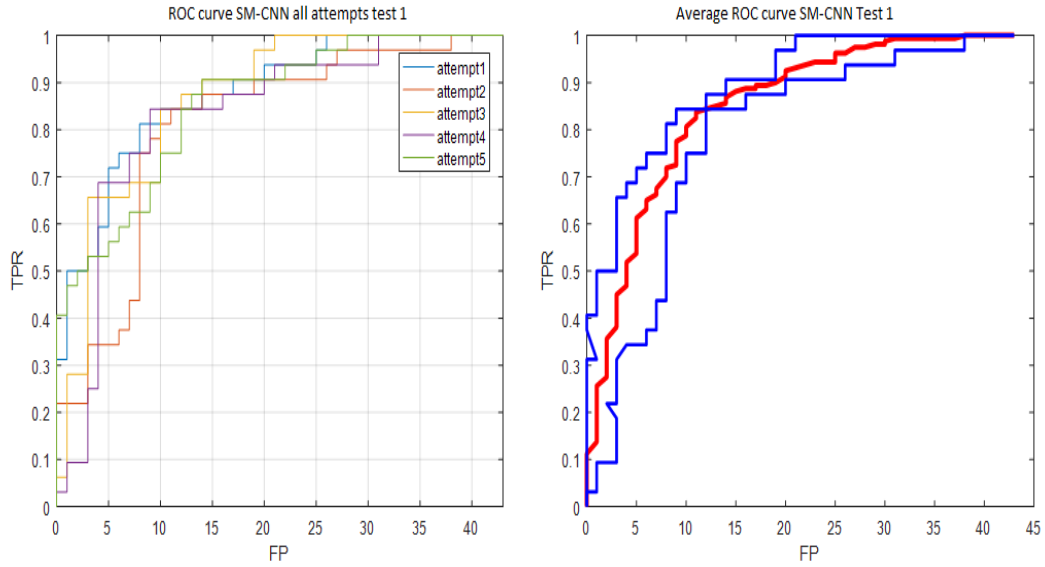


Figure4.19 SM-CNN ROC curves on test data1. On the left, each color represents one of the five runs. On the right, the vertical averaging ROC curve is plotted in red and the envelope in blue

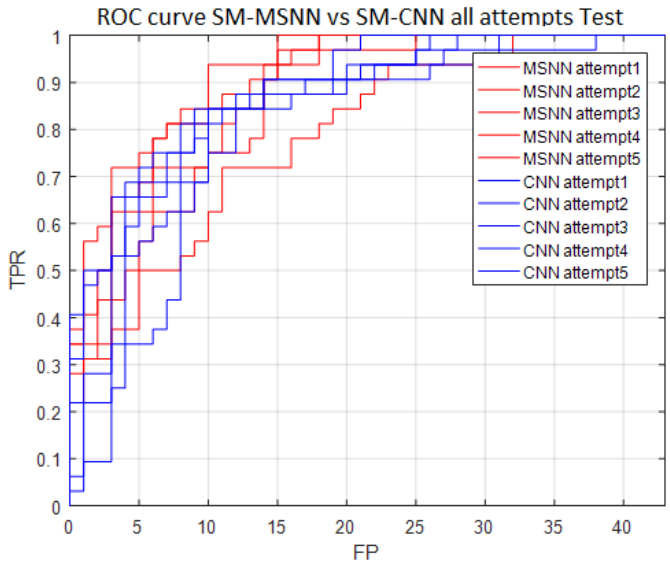


Figure 4.20 All ROC curves of test data1 for the five runs with SM-MSNN in red and SM-CNN in blue

The vertical averaging ROC curves in Figure 4.21 exhibit a very close performance with a slight advantage seen for the MS-MSNN algorithm. This same phenomenon is observed with the cumulative ROC curve; however, the difference in performance is even smaller.

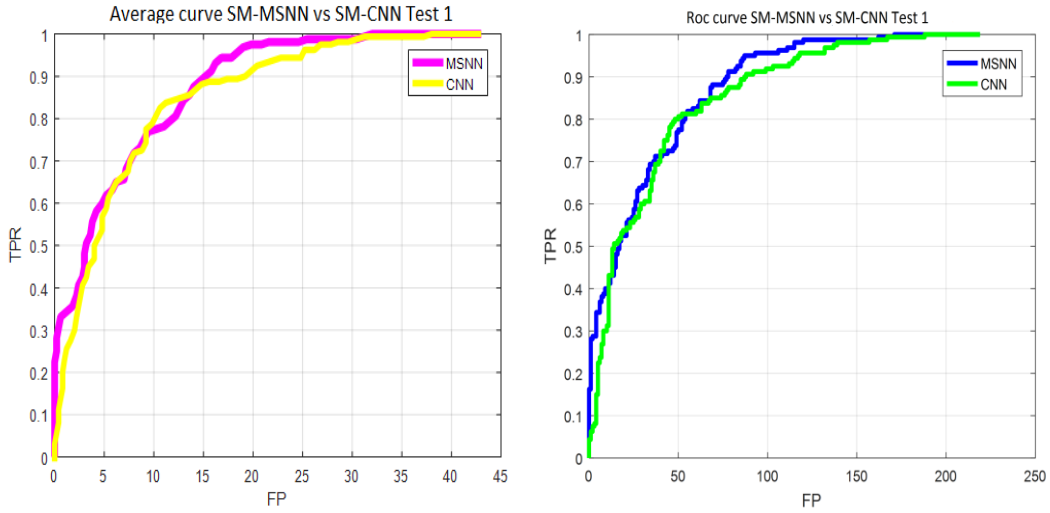


Figure 4.21 SM-MSNN and SM-CNN average ROC curve comparison on test data1.
On the left, vertical averaging shows SM-MSNN in pink and SM-CNN in yellow.
On the right, the cumulative ROC curve shows SM-MSNN as blue and SM-CNN as green.

Plotting the vertical average curves together with the cumulative ROC curve in Figure 4.22 shows a very close performance between the SM-MSNN and SM-CNN algorithms.

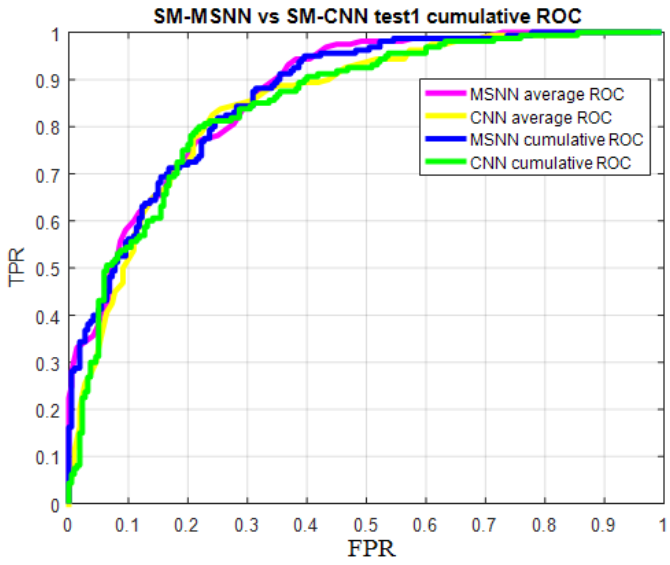


Figure 4.22 Results for test data1 show the SM-MSNN vertical averaging curve as pink. The SM-MSNN cumulative ROC curve is blue. The SM-CNN vertical averaging curve is yellow. The SM-CNN cumulative ROC curve is green.

We see in test data1, SM-MSNN and SM-CNN are very similar in the way each classifies images (bags) where images with a target are positive and images with no target are negative.

To further investigate the performance of the two algorithms, a second test using a different dataset was run. This dataset is made up of 127 aerial images of the same size as the images in the training dataset and test data1; among them, 39 are positive and 88 are negative. In the positive examples, we have 29 images of a parking lot as shown in Figure 4.23(c)–(f), where many vehicles are parked close to each other. The other 10 images may have one or more vehicles but they are more sparsely grouped as shown in Figure 4.23(a) and (b). The negative images are selected with no target in them and they look like the images shown in Figure 4.3.

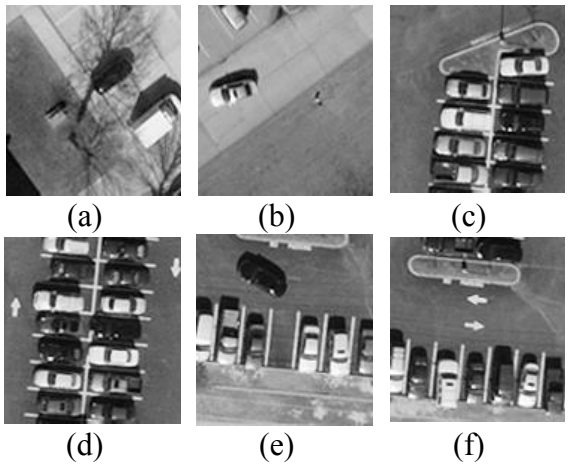


Figure 4.23 Sample of positive test images from dataset 2

The same tests are performed and the ROC curves are plotted. The ROC curves for the five runs of SM-MSNN and their vertical averaging curve are displayed in Figure 4.24. Figure 4.25 displays the same curves for SM-CNN. For the SM-MSNN most of the runs have a close performance. This reflects on the envelop of the vertical averaging curve making the SM-MSNN look spread out. On the other hand, we see the runs of the SM-CNN following the same pattern with some minor differences. While this makes the SM-CNN looks more compact, we can say that this is the best SM-CNN can do.

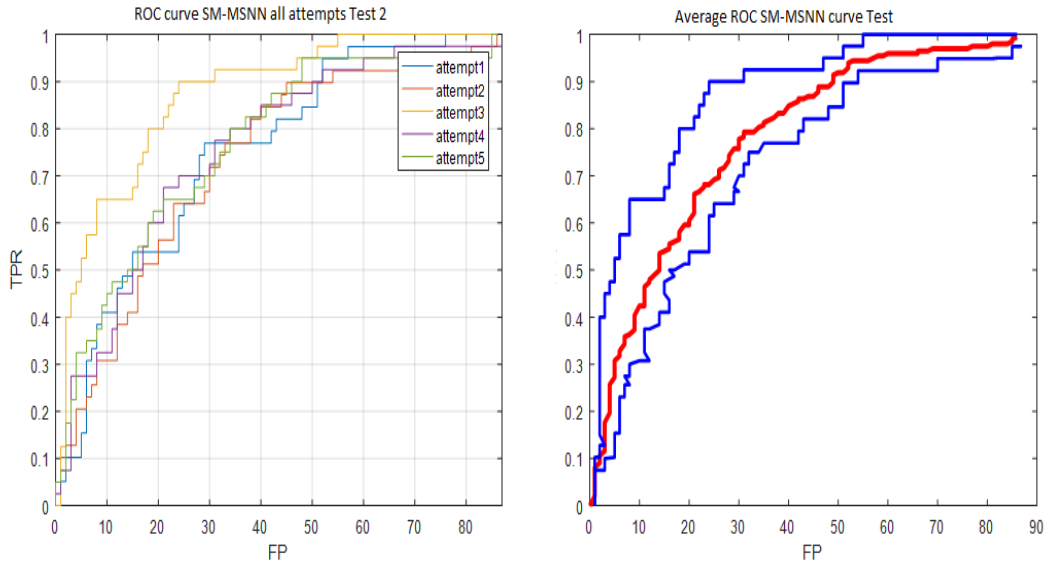


Figure 4.24 SM-MSNN ROC curves on test data2. On the left, each color represents one of the five runs. On the right the vertical averaging ROC curve is plotted in red, and the envelope in blue

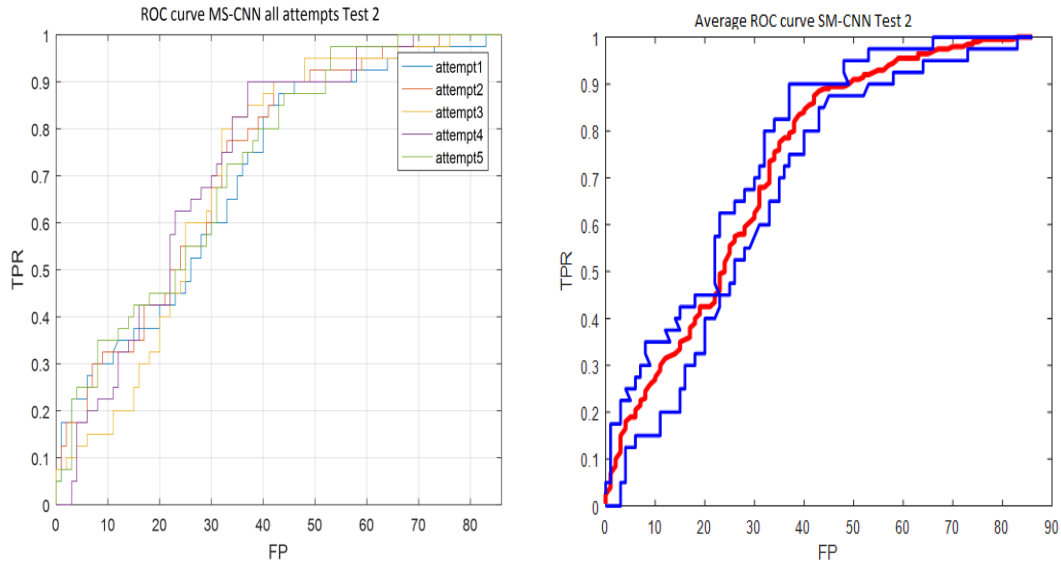


Figure 4.25 SM-CNN ROC curves on test data2. On the left, each color represents one of the five runs. On the right, the vertical averaging ROC curve is plotted in red and the envelope in blue

Looking at the ROC curves of all five runs for both algorithms in Figure 4.26, SM-MSNN has an edge over SM-CNN up to TPR = 75%; then, their performance becomes similar—except for one of SM-MSNN ROC curves, which clearly outperforms all the other curves. This curve can be perceived as an outlier. It also confirms that SM-MSNN has the potential to perform better than SM-CNN.

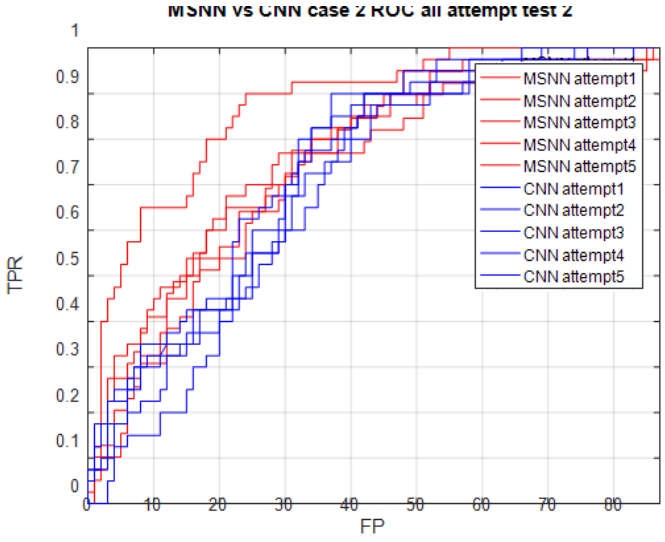


Figure 4.26 All the ROC curves of the test data2 for the five runs: The SM-MSNN lines are red and the SM-CNN lines are blue.

Figure 4.27 shows that both averaging methods yield the same conclusion. SM-MSNN outperforms SM-CNN in vertical averaging and in cumulative ROC curve. Figure 4.28 shows that the two curves for SM-MSNN are nearly identical. The same can be said about the two curves for the SM-CNN.

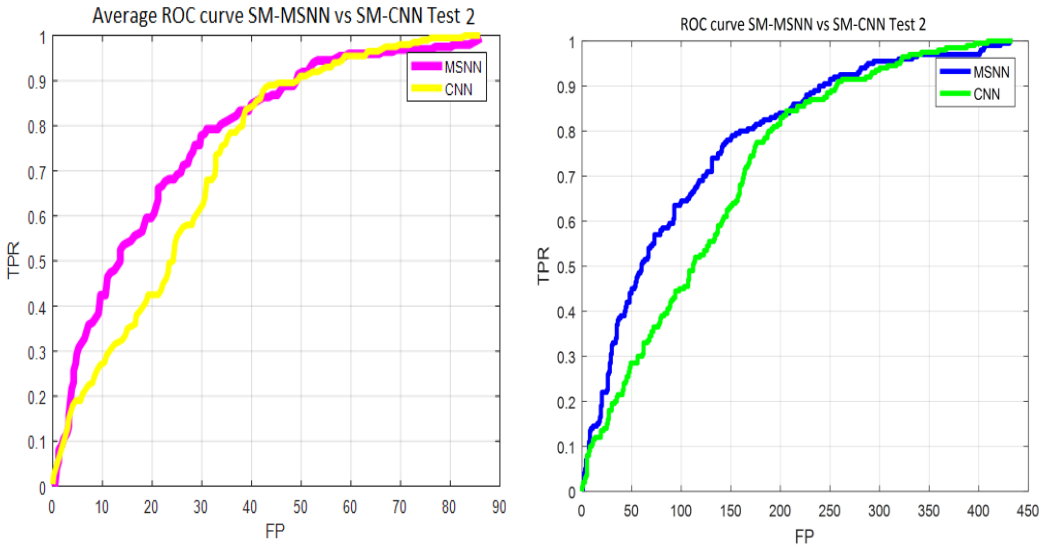


Figure 4.27 SM-MSNN and SM-CNN average ROC curve comparison on test data2. On the left, vertical averaging, SM-MSNN is pink, SM-CNN is yellow. On the right, the cumulative ROC curve, i.e., SM-MSNN is blue and SM-CNN is green.

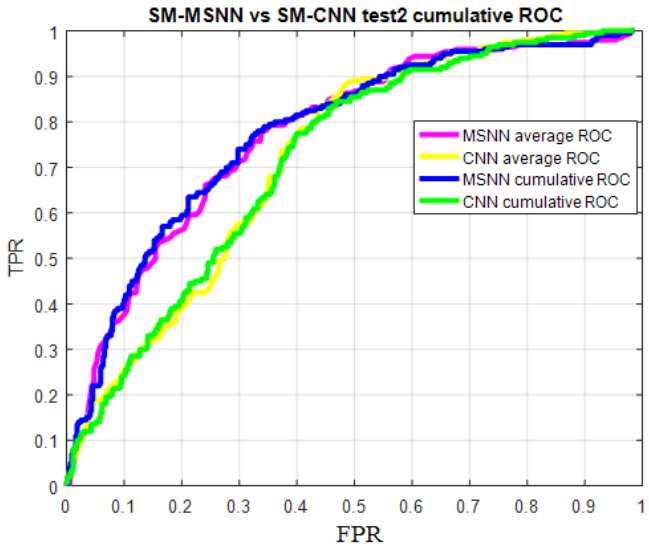


Figure 4.28 Results for test data2. The SM-MSNN vertical averaging curve is pink. The SM-MSNN cumulative ROC curve is blue. The SM-CNN vertical averaging curve is yellow. The SM-CNN cumulative ROC curve is green.

4.2.1.2 RSM-MSNN vs. RSM-CNN

We reviewed the results obtained from the SM case. Now we look at the results from RSM-MSNN versus RSM-CNN. The RSM case is when the back propagated instance is randomly selected from a list of n instances. These n instances are the top instances among the N instances of a bag. This is applied to both positive and negative bags. The same datasets used for training and testing in the SM case will be used in the RSM case.

The resulting ROC curves for the five runs of the RSM-MSNN algorithm on the training data and their vertical average curve are shown in Figure 4.29. The average ROC curve is compact for the RSM-MSNN indicating a close performance. In the vertical averaging curve, we see no case in which RSM-MSNN achieves perfect detection with no false alarms.

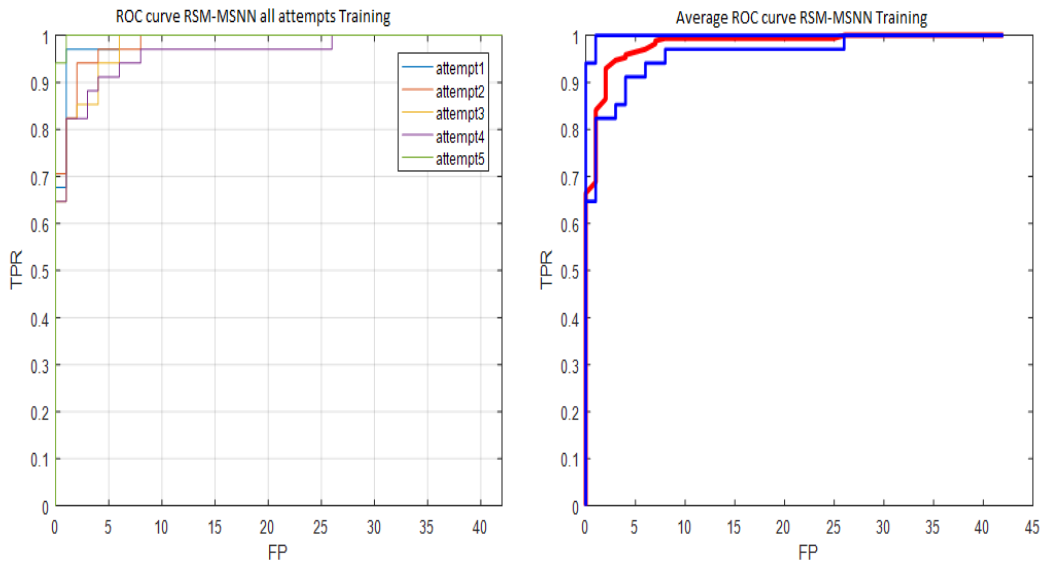


Figure 4.29 RSM-MSNN ROC curves on training data. On the left, each color represents one of the five runs. On the right, the vertical averaging ROC curve is plotted in red and the envelope in blue.

The resulting ROC curves for the five runs of the RSM-CNN algorithm on the training data and their vertical average curve are shown in Figure 4.30. The average ROC curve seems less compact compared to RSM-MSNN, but by looking at all five runs of the RSM-CNN, we can see that the third run is the one dragging the envelop down. If that run is ignored the RSM-CNN will have a more compact vertical averaging curve.

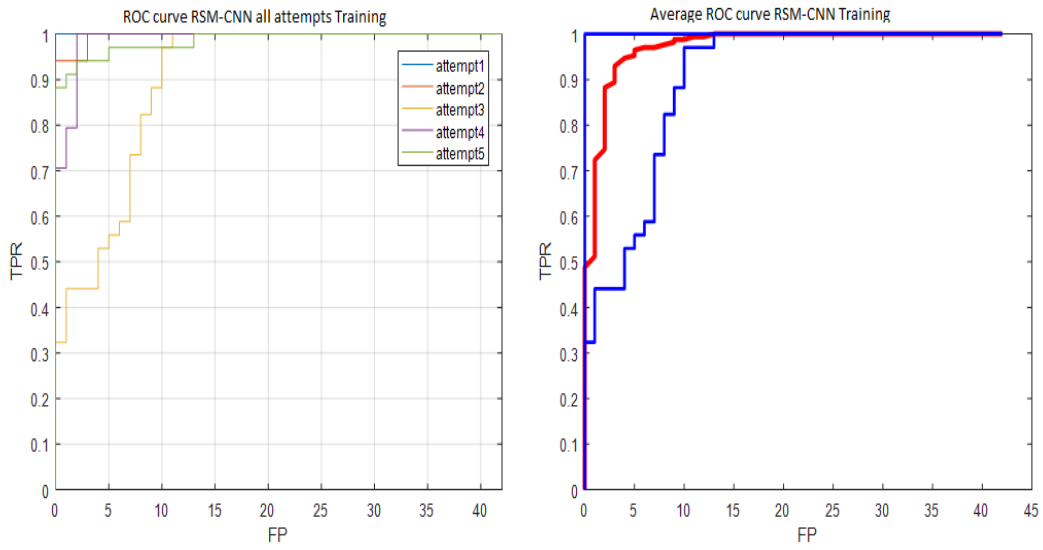


Figure 4.30 RSM-CNN ROC curves on training data. On the left, each color represents one of the five runs. On the right the vertical averaging ROC curve is plotted in red, and the envelope in blue.

Plotting the curves for the five runs for both algorithms, Figure 4.31 shows a close performance among all curves except for one curve for one of RSM-CNN runs.

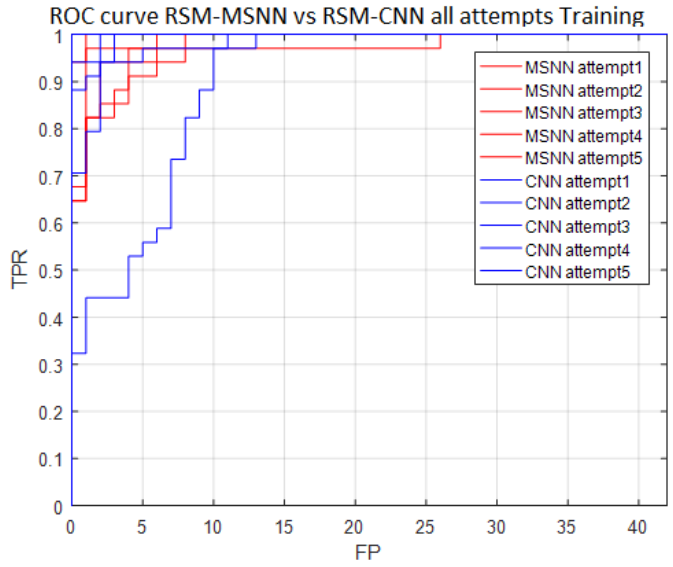


Figure4.31 All the ROC curves of the training data for the five runs are shown as RSM-MSNN (red) and RSM-CNN (blue)

Still, Figure 4.32 shows that the two vertical average curves are nearly on top of each other with a slight advantage for RSM-MSNN. The cumulative ROC curve shows RSM-MSNN as having a better performance than RSM-CNN—up to TPR=50%. Then, the two algorithms exchange the lead before meeting at TPR=100% with the same number of false positives.

Comparing the vertical average curves and the cumulative curves, Figure 4.33 shows a big difference between the vertical average curve and the cumulative ROC curve for each algorithm. However, they still tell the same story, that is, the performance of both algorithms is very close with a slight advantage for RSM-MSNN.

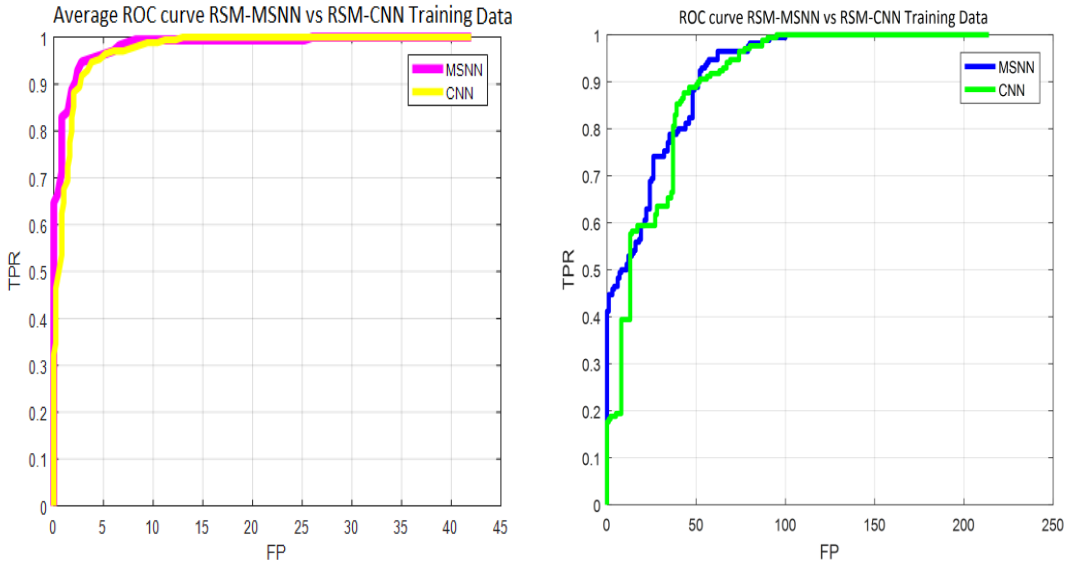


Figure 4.32 RSM-MSNN and RSM-CNN average ROC curve comparison on training data. On the left, vertical averaging RSM-MSNN curve is pink, and the RSM-CNN curve is yellow. On the right, the cumulative ROC curve RSM-MSNN is blue, and the RSM-CNN curve is green

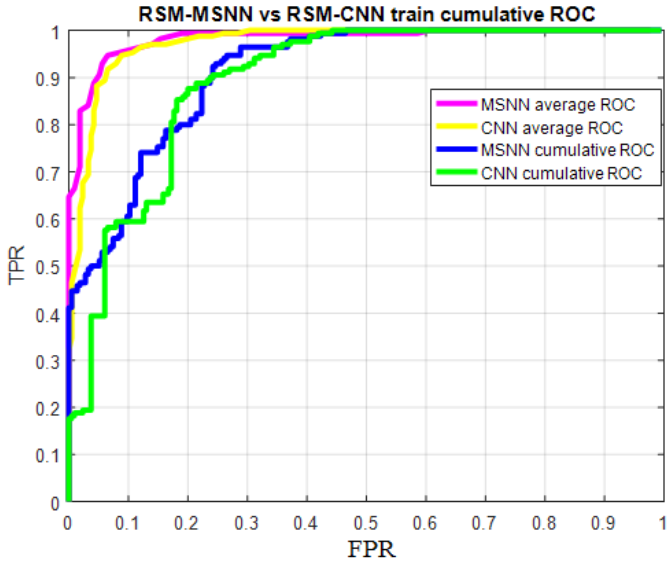


Figure 4.33 Results for training data shows RSM-MSNN vertical averaging curve as pink and the RSM-MSNN cumulative ROC curve as blue. The RSM-CNN vertical averaging curve is shown here as yellow, while the RSM-CNN cumulative ROC curve is green.

The performance of RSM-MSNN and RSM-CNN is tested using test data1. The five ROC curves for the RSM-MSNN are shown Figure 4.34, and the vertical average

curve is also in the same figure. The envelope is small at low values, but it gets bigger as the TPR value increases.

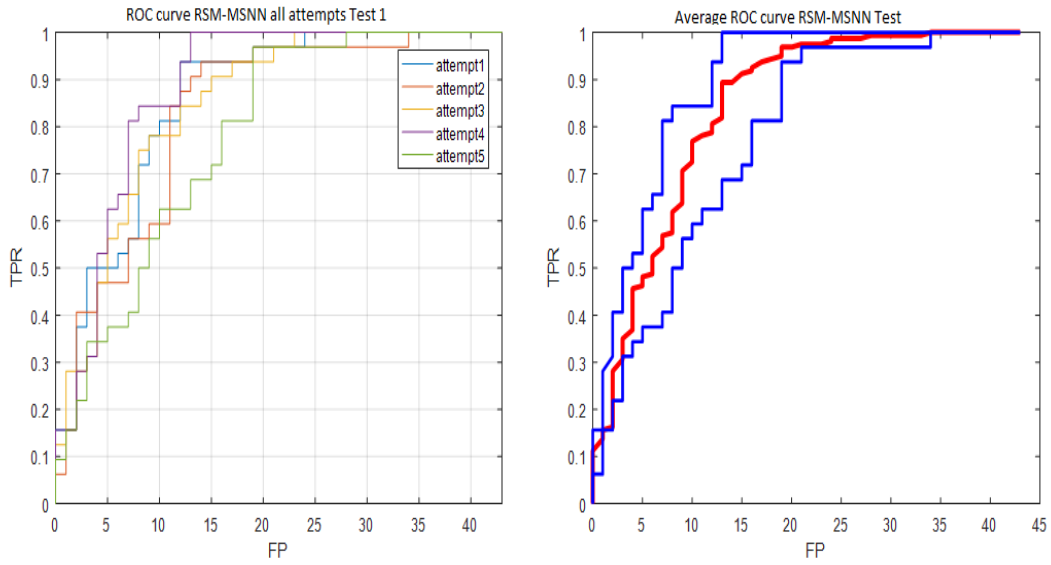


Figure 4.34 RSM-MSNN ROC curves on test data1. On the left, each color represents one of the five runs. On the right, the vertical averaging ROC curve is plotted in red and the envelope in blue

For the RSM-CNN, Figure 4.35 shows the five-run test results. The vertical averaging curve with its envelope is also shown. The five ROC curves seem to follow a similar pattern except for attempt 1 curve in blue. This can be attributed to the different initialization used for each curve. This can also be perceived in the vertical average curve where the envelope seems to form a uniform shape around the average curve. This shape is only disturbed by the attempt 1 curve.

When comparing the five runs for both algorithms in Figure 4.36 we see most of the RSM-MSNN curves dominating the RSM-CNN curves except for one curve.

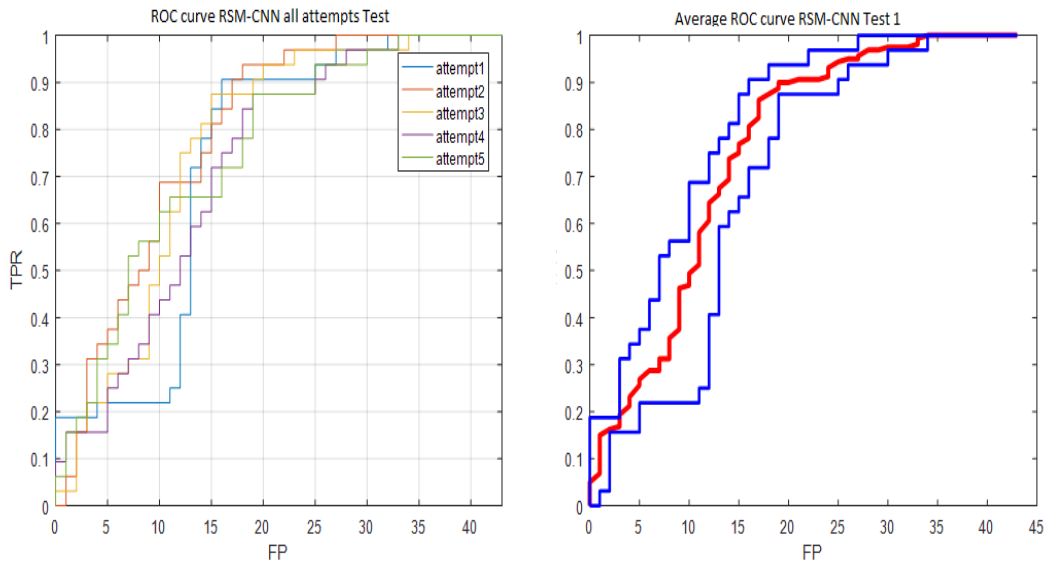


Figure 4.35 RSM-CNN ROC curves on test data1. On the left, each color represents one of the five runs. On the right, the vertical averaging ROC curve is plotted in red and the envelope in blue

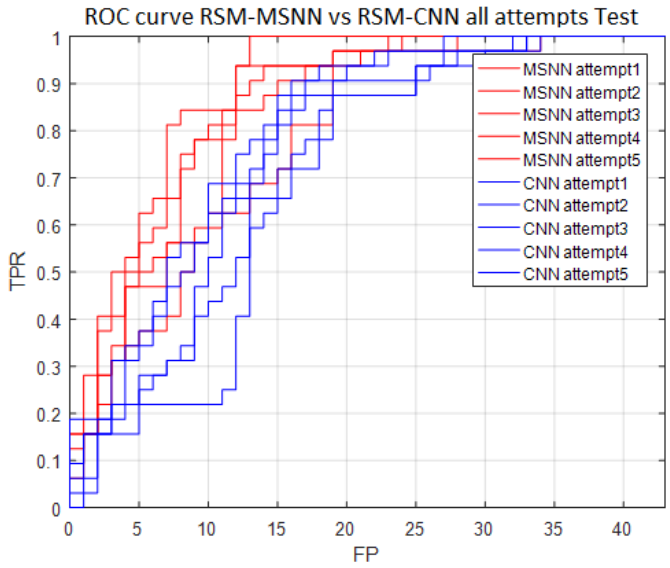


Figure 4.36 All five runs for RSM-MSNN and RSM-CNN test data1

This is confirmed in Figure 4.37 by both vertical averaging curves and cumulative ROC curves. However, the cumulative ROC curve shows a small phase where RSM-CNN takes the lead but RSM-MSNN recovers after that. This phenomenon can be related to the one case of the five runs where RSM-MSNN is inferior to the RSM-CNN. The vertical averaging curve tells us which of the two algorithms dominates the other in general, while

cumulative ROC curve, which utilizes results from all five runs, provides more information; it informs us that there is a phase where RSM-CNN is better than RSM-MSNN.

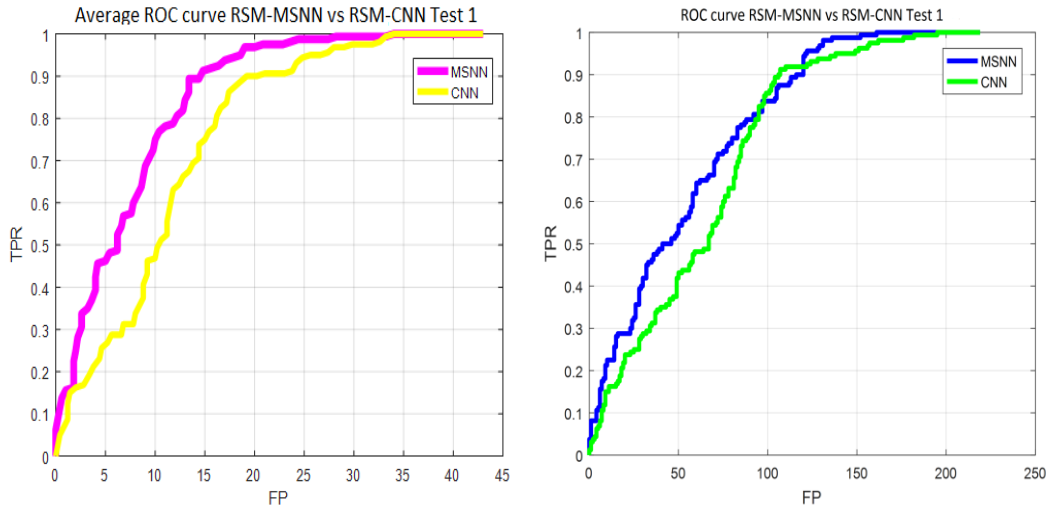


Figure 4.37 RSM-MSNN and RSM-CNN average curve comparison. to the left vertical averaging. to the right cumulative curve test data1

By plotting both types of average curves for the two algorithms in Figure 4.38, we see a clear superiority for the RSM-MSNN despite some small setbacks.

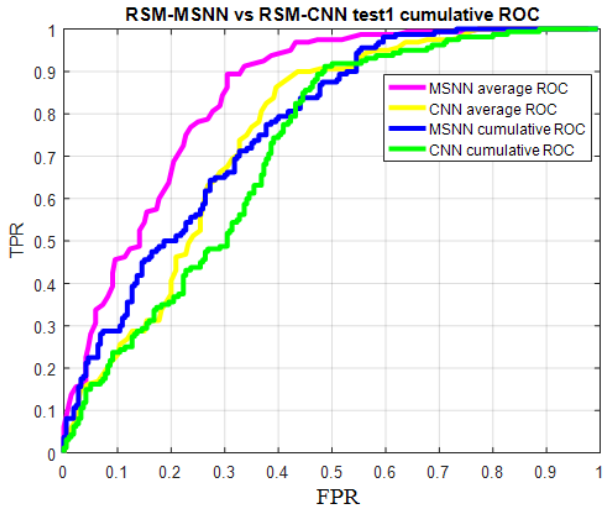


Figure 4.38 Vertical average curves and cumulative ROC curves for RSM-MSNN and RSM-CNN, test data1

Similar to the SM case, we turn to test dataset2 to try and find which of the two algorithms dominate. Figure 4.39 shows a lot of similarity between the RSM-MSNN ROC curves with small differences. The vertical average curve confirms this conclusion by displaying a more or less uniform envelope around the curve.

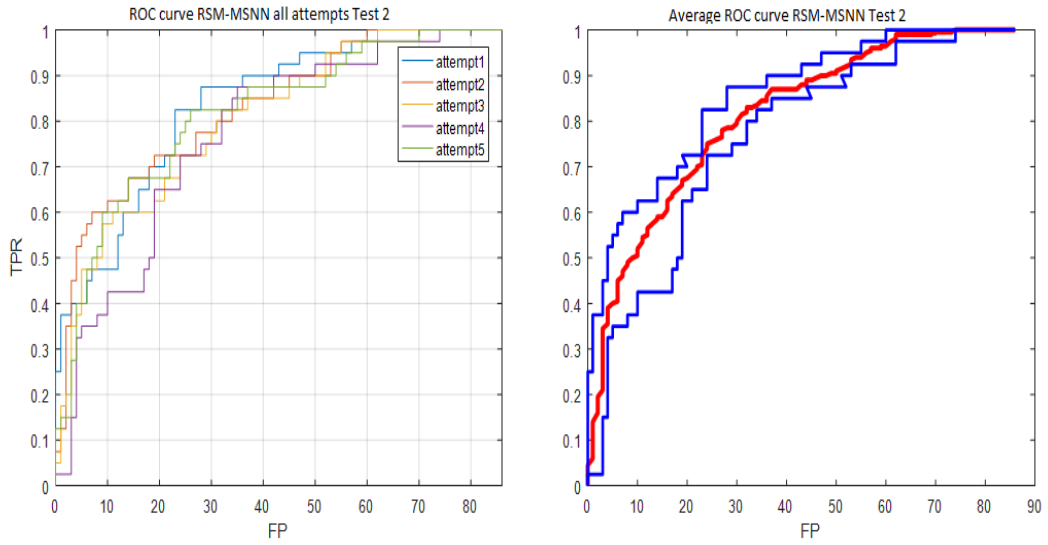


Figure 4.39 RSM-MSNN ROC curves on test data2. On the left, each color represents one of the five runs. On the right the vertical averaging ROC curve is plotted in red and the envelope is plotted in blue

The RSM-CNN ROC curves in Figure 4.40 shows some uniformity too. This is also confirmed by the vertical average curve in the same figure. However, the RSM-CNN envelope is more spread out compared to the RSM-MSNN.

Test dataset 2 again decides in favor of MSNN over CNN. The comparison of the five runs for both algorithms in Figure 4.41 shows an undeniable superiority for the RSM-MSNN over the RSM-CNN. All the red curves representing the five curves of the RSM-MSNN are above the blue curves representing the RSM-CNN. Also, the red curves are compact indicating a solid performance, whereas the blue curves are more spread out.

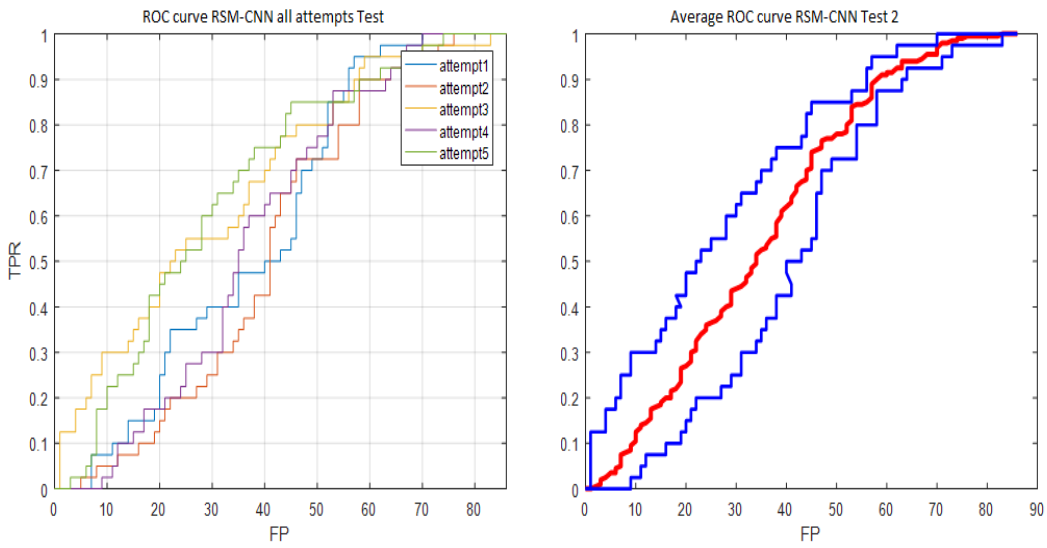


Figure 4.40 RSM-CNN ROC curves on test data2. On the left, each color represents one of the five runs. On the right the vertical averaging ROC curve is plotted in red and the envelope is plotted in blue

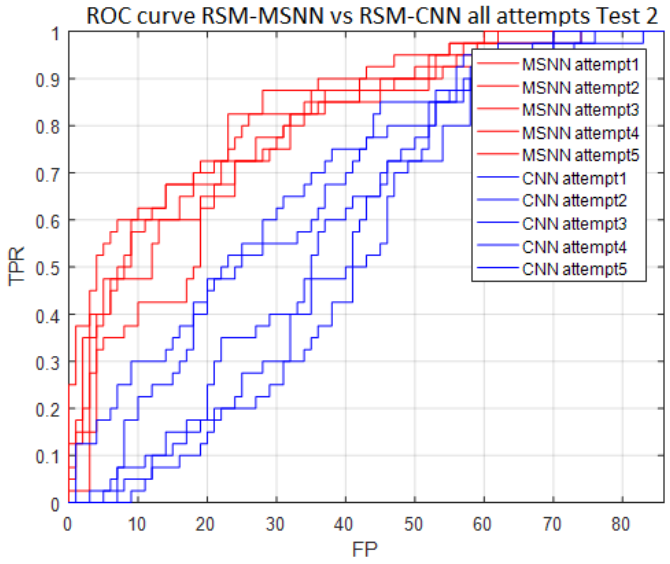


Figure 4.41 All five runs for RSM-MSNN and RSM-CNN from test data2

The observation made above is confirmed by both the vertical averaging curve and the cumulative ROC curve in Figure 4.42. The RSM-MSNN clearly dominates the RSM-CNN by a big margin.

Putting all the average curves together in Figure 4.43 we see some differences between the vertical averaging curve and cumulative ROC curve for the RSM-MSNN. However, this is not significant since RSM-MSNN maintains its lead over the RSM-CNN.

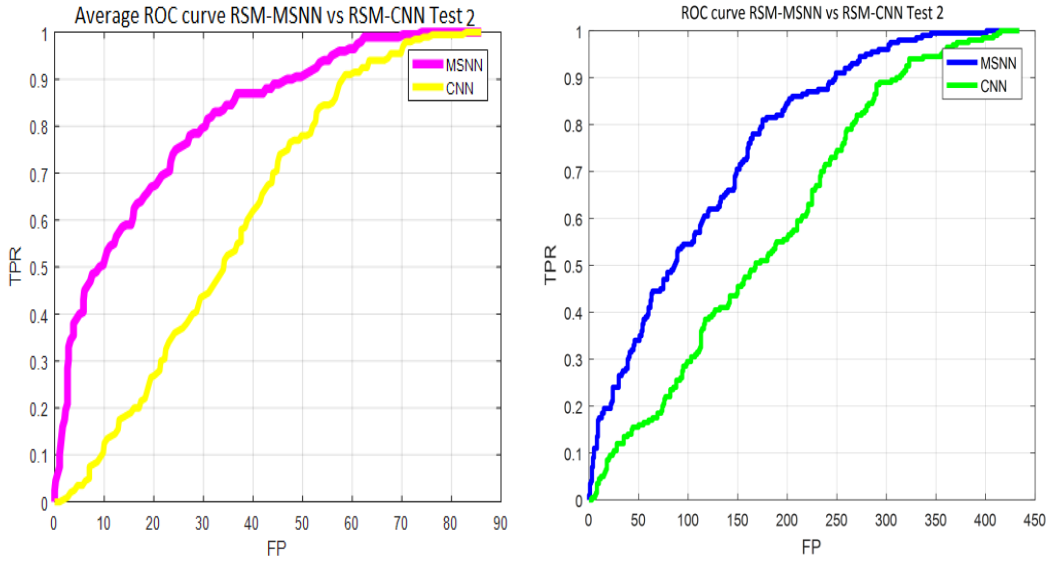


Figure 4.42 RSM-MSNN and RSM-CNN average curve comparing vertical averaging on the left to the cumulative curve on the right based on test data2

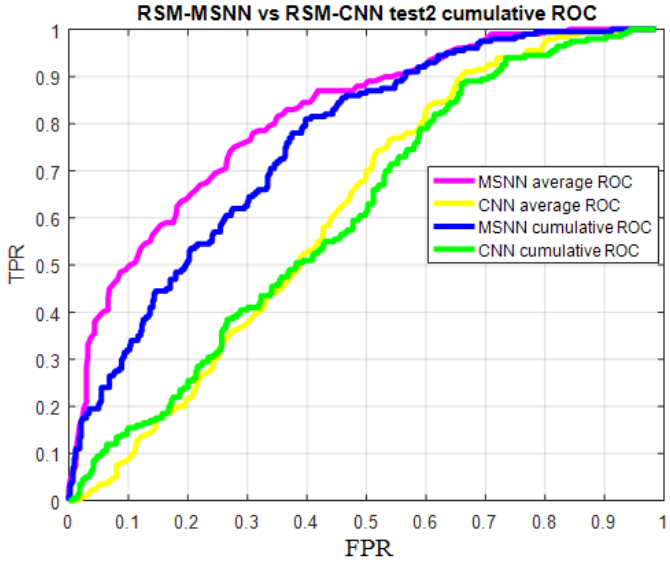


Figure 4.43 Vertical average curves and cumulative ROC curves for RSM-MSNN and RSM-CNN from test data2

We have seen the ROC curves for SM-MSNN and RSM-MSNN. Both algorithms were compared with a corresponding CNN version. The test data1 which bears a lot of similarity to the training dataset put the SM-MSNN and the SM-CNN on equal grounds. Test data2, which has many images with cars arranged in parking lots, gives some edge to the SM-MSNN over SM-CNN.

The ROC curves for RSM-MSNN and RSM-CNN were more conclusive about the difference in performance especially when test data2 was used. A clear advantage of RSM-MSNN over RSM-CNN is noted.

All the ROC curves presented in this study classify images (bags) as either target (positive) or non-target (negative). However, more clarity about what part of the image that triggered the decision is needed. In other words, in the case of an image determined as target, we do not know if the detection was triggered by a positive instance in the image (bag) or by a negative instance. The images fed into the algorithm during the training process did not have any specification of the location of the target. The instances were raw sub-images that only roughly indicated dimensions that would fit target objectives. We relied on the algorithm's ability to find and determine the target.

We need to confirm that it was the target that triggered the detection of a positive image and it was its absence that classified an image (bag) as negative. We need to investigate if our algorithms are detecting the target(s) in each image.

4.2.2 Detection (instance classification)

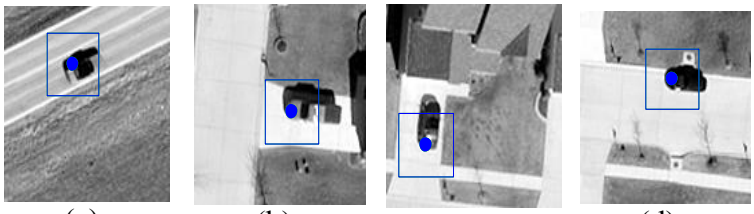
Since the datasets we are using for training and testing contain a considerable number of images, we needed a way to evaluate the instance classification performance of our algorithms. Usually ROC curves are used in these situations to evaluate the

performance of the algorithms. However, due to the shape of our target (the vehicle), there is a genuine concern that the ROC curve may not reflect actual results. We use the center of the target to plot the ROC curve. This method makes sense in the supervised learning framework, because the algorithm is trained using the center of the target. However, the algorithm in the MIL framework does not learn the center of the target. Rather, it looks for the optimal instance that contains the target. A positive bag contains at least one positive instance, but it may contain more. We do not know how many positive instances there are, and we do not know which among them is the optimal instance. Instead, we use tables where we count the number of detected targets versus non-detected at a fixed threshold. We have chosen two thresholds we believe are best to assess the performance of our algorithms. The thresholds are 0.5 and 0.9. The threshold 0.5 represent the minimum performance required by each of our algorithms. Whereas 0.9 is to see how many times our algorithm was able to push target confidences close to 1.

The number of false alarms in a positive image and the number of false alarms in a negative image are counted separately. The reason is that each type of false alarm has a different meaning. A false alarm in a positive image may not disturb the correct classification of the image (bag), but it helps us measure the algorithm's ability to properly distinguish the target from the background. A false alarm in a negative image indicates that the algorithm will misclassify a negative image (bag) as positive. Therefore, it's an indication that our algorithm may have failed to operate within the multiple-instance learning framework.

To quantify the detection and make it easier to compare the algorithms we use these terms for our tables:

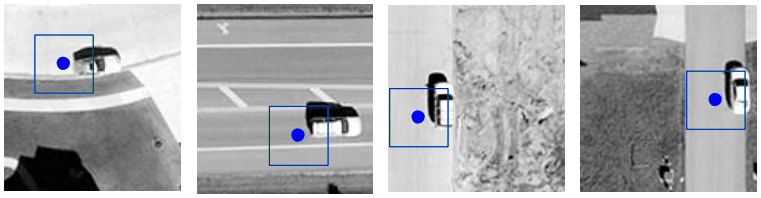
- Maximum detection on target (MDT): If the maximum confidence is on the vehicle or one of the vehicles in the image as shown in Figure 4.44.



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

Figure 4.44 Maximum confidence (blue point) on target

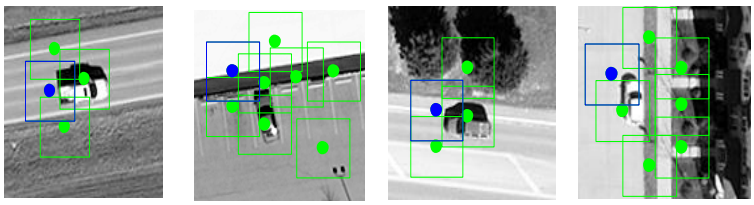
- Partial maximum detection on target (PMDT): If a part of the vehicle only appears in the instance marked by the blue box but not the whole vehicle as shown in Figure 4.45.



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

Figure 4.45 instance (blue box) contains a part of the target

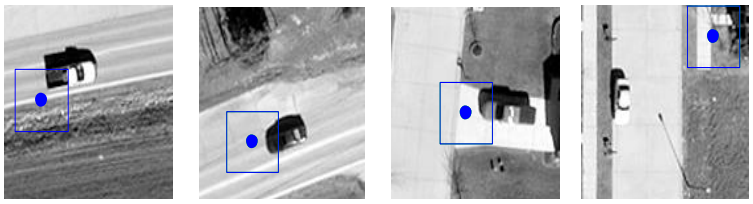
- Non-maximum detection on target (NMDT): If the target produces a confidence higher than the threshold, but it is not the maximum confidence in the image as shown in Figure 4.46.



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

Figure 4.46 Confidence higher than threshold on target (green point) but not the maximum (blue dot)

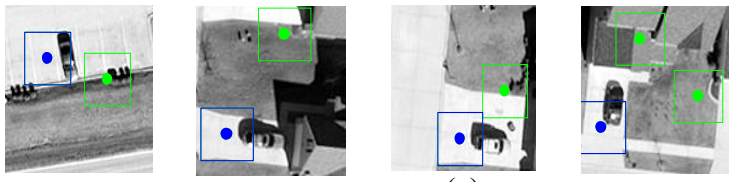
- Non-target detection (NDT): If there is a confidence higher than the threshold in the image but not produced by the target as shown in Figure 4.47.



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

Figure 4.47 Maximum confidence higher than the threshold but not on target (blue dot)

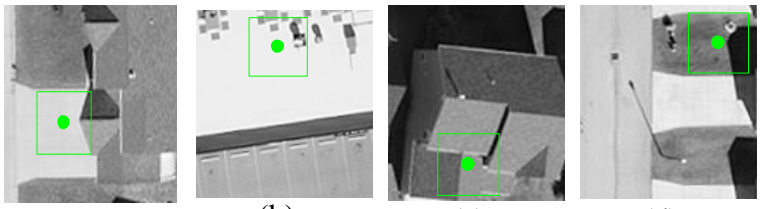
- PBFA: Number of false alarms in the positive image as shown in Figure 4.48.



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

Figure 4.48 False alarm in a positive image, green dot **not** on a vehicle.

- NBFA: Number of false alarms in the negative image as shown in Figure 4.49.



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

Figure 4.49 The presence of a green dot in a negative image indicates a false alarm

It is important to note that these detections (dots) are a bit different from the detections we saw in the supervised case. In the supervised case, the algorithm looks for the center of the target. Then a bounding box is drawn around the target. In the MIL case, the algorithm looks for an instance with the same size as the target that is likely to have the target in it. Then, the center of the instance is considered the center of the target.

To better understand the values obtained in the table, two charts are plotted—a detection scores chart and a false alarms statistics chart. The former depicts the four possible cases of detection MDT in blue, NMDT in orange, NDT in gray and PMDT in yellow. The total of the four scores equals the number of positive images in the test set; thus, a column representation is used. MDT is the best detection case and its blue color

shows that it is the best algorithm. If the whole column is blue, then we have an excellent detection.

The false alarms statistics chart gives the other important part of the evaluation, that is how many false alarms were generated by an algorithm. There are two colors, green for false alarms in the positive bag, and blue for false alarms in negative bag. The best algorithm is the one with the lowest values.

4.2.2.1 Experiment 1

In this experiment the bags are generated similarly to the bag level classification.

4.2.2.1.1 Training data: 34 positive images, 43 negative images

The results for testing the algorithms on the training dataset are summarized in Table 4.3. The first row is the threshold used to evaluate the algorithms. We have two thresholds, 0.5 and 0.9. The next four rows (MDT, NMDT, NDT and PMDT) are related to target detection. The total of these four values equals the number of positive images. These detection scores are displayed in a chart as shown in Figure 4.50. An algorithm's performance is optimal if MDT is equal to the total number of positive images and NMDT and NDT are equal to zero. Images classified as PMDT are not optimal but they are still considered good results. The last two rows (PBFA and NBFA) shows the number of false alarms. They are displayed in a separate chart such as the one shown in Figure 4.51. These values are shown in the table in a ratio form. For PBFA, the value in the denominator is equal to the number of positive images, whereas the denominator in NBFA is equal to the number of negative images. We only use the values in the numerator for the chart. An algorithm's performance is optimal if both these values are equal to zero.

If we look at the first column in Figure 4.50 (SM-MSNN, threshold=0.5), MDT is equal to nine, NMDT is equal to one, NDT is equal to two and PMDT is equal to 24. If we only look at MDT score, then this algorithm has a poor performance. However, when combining the scores of MDT and PMDT, this algorithm's performance improves significantly. In addition to that, PBFA is 14/34, which means that we have less than one false alarm per two positive images. On the other hand, NBFA is 4/43. This algorithm generated very few false alarms in the negative images.

The first thing we notice is that, the scores are barely affected by the change in threshold. When we look at the MDT scores only, we see that all the algorithms detect the vehicle in around 60% of the images except for SM-MSNN which had a lower detection rate of 27%. However, looking at MDT and PMDT scores together, SM-MSNN takes the lead over all the other algorithms and detects the vehicle in more than 94% of the images. given that we are working in the MIL framework, it is a stretch to expect the instance with the highest threshold to always contain the full target.

Table 4.3: Experiment 1 detection results on training data for MIL-MSNN and MIL-CNN, thresholds 0.5 and 0.9

	SM-MSNN		**SM-CNN**		**RSM-MSNN**		**RSM-CNN**	
threshold	0.5	0.9	0.5	0.9	0.5	0.9	0.5	0.9
MDT	9	9	20	20	20	20	18	18
NMDT	0	0	2	2	6	6	12	12
NDT	2	2	5	5	0	0	2	2
PMDT	23	23	7	7	8	8	2	2
PBFA	14/34	11/34	10/34	8/34	172/34	135/34	142/34	118/34
NBFA	4/43	1/43	14/43	6/43	152/43	95/43	59/43	26/43

The false alarms statistics chart is in Figure 4.51. Unlike the detection scores chart, the selected threshold influences the number of false alarms. The SM-MSNN and SM-CNN are in the lead with the lowest number of false alarms in the positive images. SM-

MSNN has the lowest number of false alarms in the negative images. Both "RSM" algorithms have a very high number of false alarms. This could be due to the way the "RSM" algorithms operate. The random roulette selection could be confusing the network instead of helping it to learn.

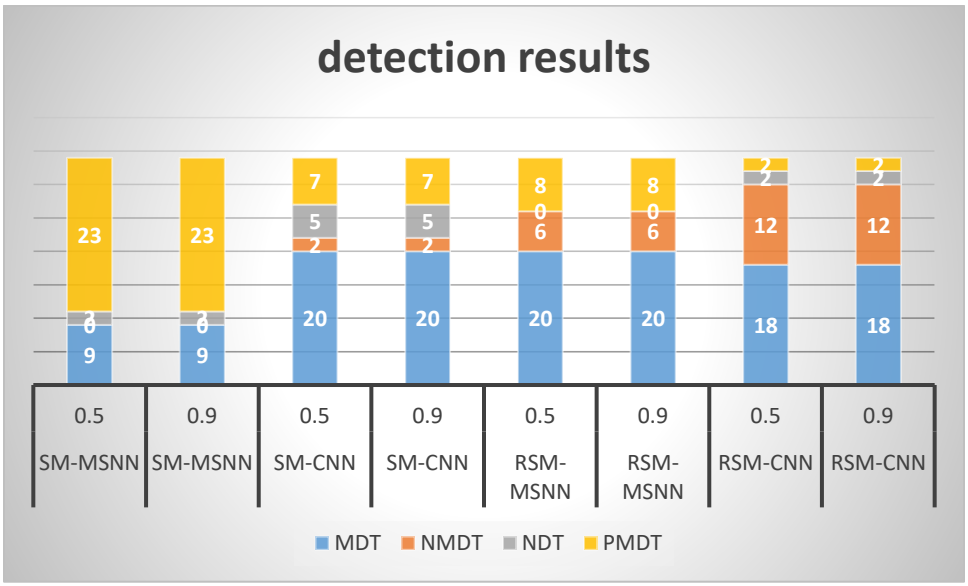


Figure 4. 50 Experiment 1 detection scores in training data for all algorithms using thresholds 0.5 & 0.9. Blue is MDT (Figure 4.44), Orange is NMDT (Figure 4.46). Gray is NDT (Figure 4.47). yellow is PMDT (Figure4.45)

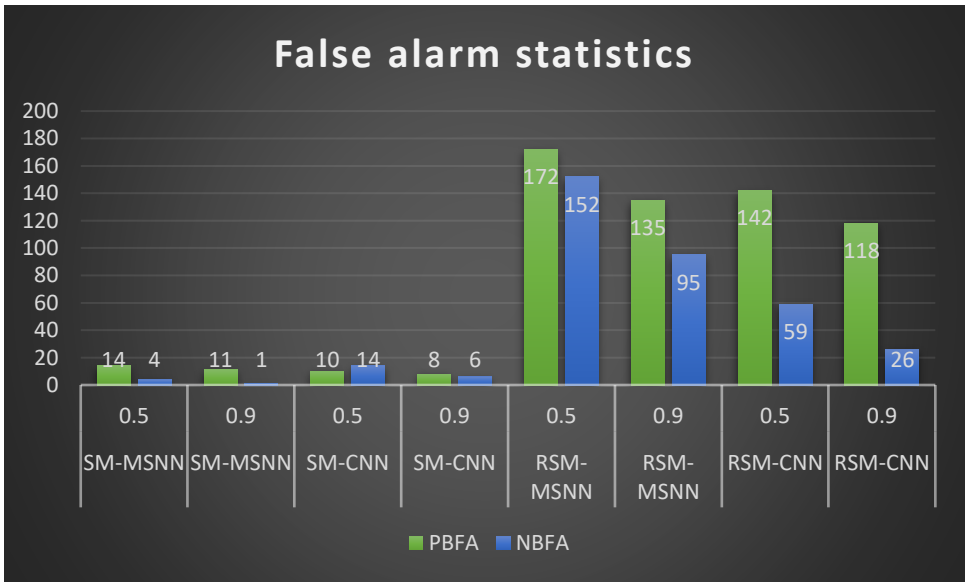


Figure 4.51 Experiment 1generated this false alarm statistics bar graph showing the training data for all algorithms using thresholds of 0.5 & 0.9. The positive bag false alarms are in green (Figure 4.48). The negative bag false alarm is in green (Figure 4.49).

The behavior of the SM-MSNN can be explained by looking at Figure 4.52. We can see the maximum confidence in area around the target but not on it. This is the case in most of the images detected as PMDT. As we explained earlier, the algorithms in the MIL framework look for the instance. Judging by the bounding box drawn around the blue dot, we can say that SM-MSNN finds an instance that has the target, but not the optimal one.

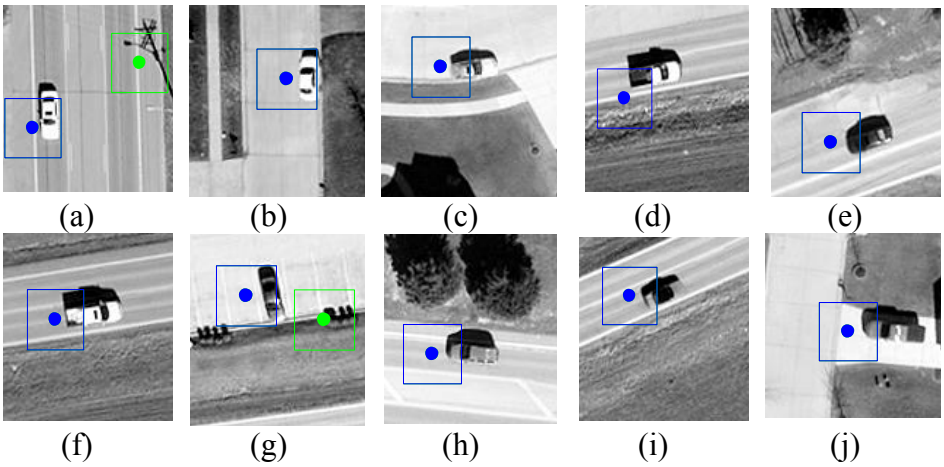


Figure4.52 SM-MSNN MDT detection samples. The blue box indicates the instance found by the target. The blue dot is its center.

4.2.2.1.2 Test data1: 32 positive images, 44 negative images.

The results for testing the algorithm on test data1 are summarized in Table 4.4. The detection scores chart and the false alarms statistics chart are shown in Figure 4.53 and Figure 4.54, respectively. Most of the remarks from the training data still hold for test data1. SM-MSNN still maintains the lead followed by SM-CNN. However, SM-MSNN as a lot of PMDT cases compared to the other algorithms. SM algorithms have a very low number of false alarms (both PBFA and NBFA) compared to RSM algorithms.

Table4.4: Experiment 1 detection results on test data1 for MIL-MSNN and MIL-CNN with threshold 0.5 and 0.9

	SM-MSNN		SM-CNN		RSM-MSNN		RSM-CNN	
Threshold	0.5	0.9	0.5	0.9	0.5	0.9	0.5	0.9
MDT	11	11	21	21	17	17	12	12
NMDT	0	0	0	0	11	11	2	2
NDT	2	2	4	4	0	0	7	7
PMDT	19	19	7	7	4	4	11	11
PBFA	6/32	3/32	5/32	4/32	135/32	83/32	52/32	41/32
NBFA	13/44	9/44	10/44	2/44	201/44	146/44	44/44	28/44

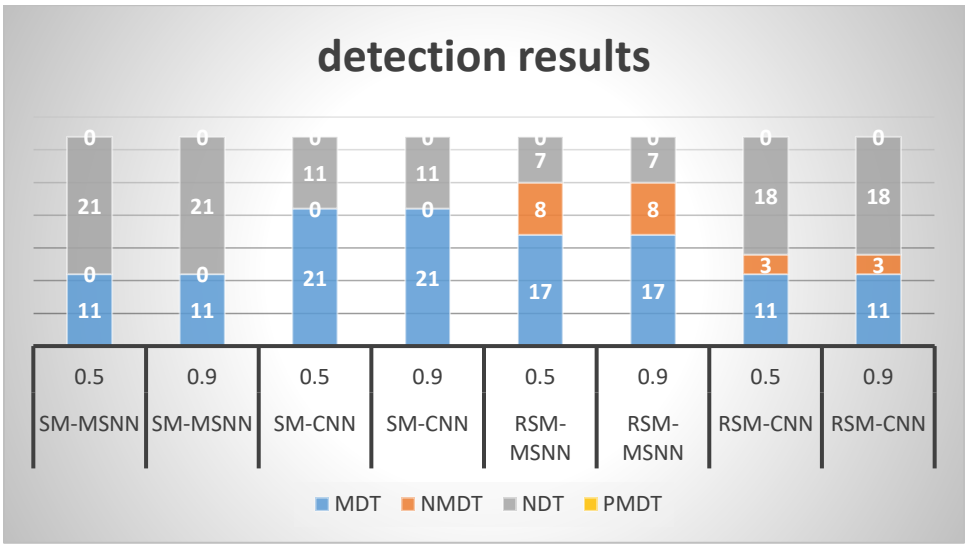


Figure 4.53 Experiment 1 detection scores in Test data1 for all algorithms using thresholds 0.5 & 0.9. Blue is MDT (Figure 4.44), Orange is NMDT (Figure4.46). Gray is NDT (Figure 4.47). Yellow is PMDT (4.45)

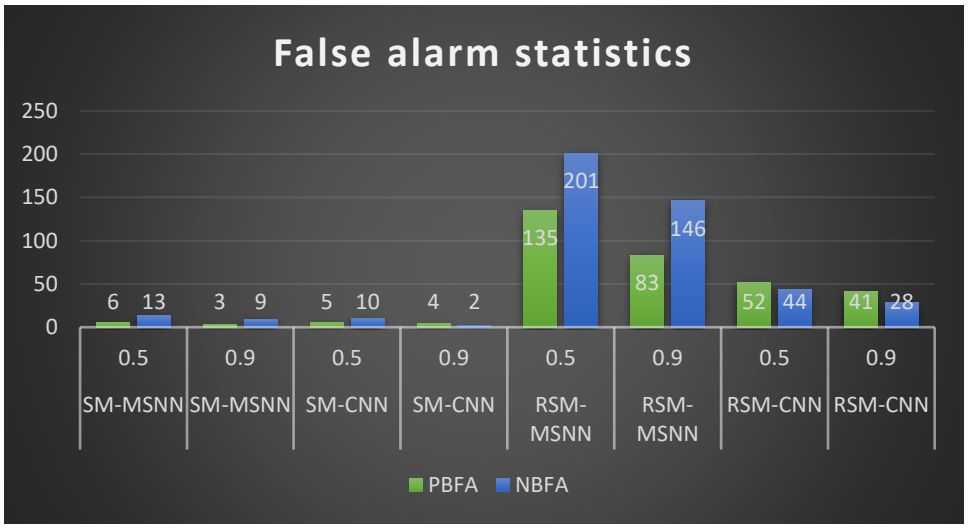


Figure 4.54 Experiment 1: False alarm statistics in test data1 for all algorithms using thresholds 0.5 & 0.9. Positive bag false alarm in green (Figure 4.49). Negative bag false alarm in green (Figure 4.49).

4.2.2.1.3 Test data2: 39 positive images, 88 negative images

The results for testing the algorithm on test data2 are summarized in Table 4.5. The detection scores chart and the false alarms statistics chart are shown in Figure 4.55 and Figure 4.56, respectively. The four algorithms have a relatively good performance. RSM algorithms take the lead this time in the MDT detection score, followed closely by SM algorithms. The SM-MSNN is in the last position, but its performance has improved compared to previous cases. When MDT and PMDT scores are considered together, all the four algorithms have a similar performance with vehicles detected in about 85% of the images. SM-MSNN is in the lead in the elimination of false alarms closely followed by the SM-CNN. RSM algorithms still generates high number of false algorithms.

Table 4.5: Experiment 1 detection results on test data2 for MIL-MSNN and MIL-CNN with threshold 0.5 and 0.9

	SM-MSNN		**SM-CNN**		**RSM-MSNN**		**RSM-CNN**	
threshold	0.5	0.9	0.5	0.9	0.5	0.9	0.5	0.9
MDT	18	18	22	22	25	25	24	24
NMDT	2	1	0	0	5	5	5	2
NDT	4	5	5	5	2	2	0	3
PMDT	15	15	12	12	7	7	10	10
PBFA	6/39	3/39	8/39	7/39	90/39	55/39	32/39	14/39
NBFA	32/88	14/39	30/88	10/88	372/88	245/88	203/88	175/88

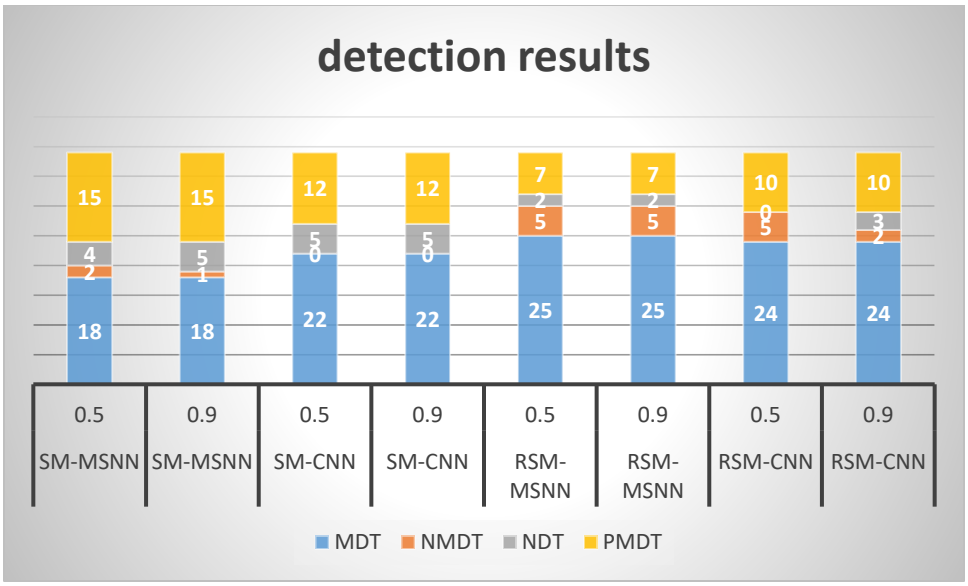


Figure 4.55 Experiment 1 detection scores in Test data2 for all algorithms using thresholds 0.5 & 0.9. Blue is MDT (Figure 4.44), Orange is NMDT (Figure 4.46). Gray is NDT (Figure 4.47). Yellow is PMDT (4.45)

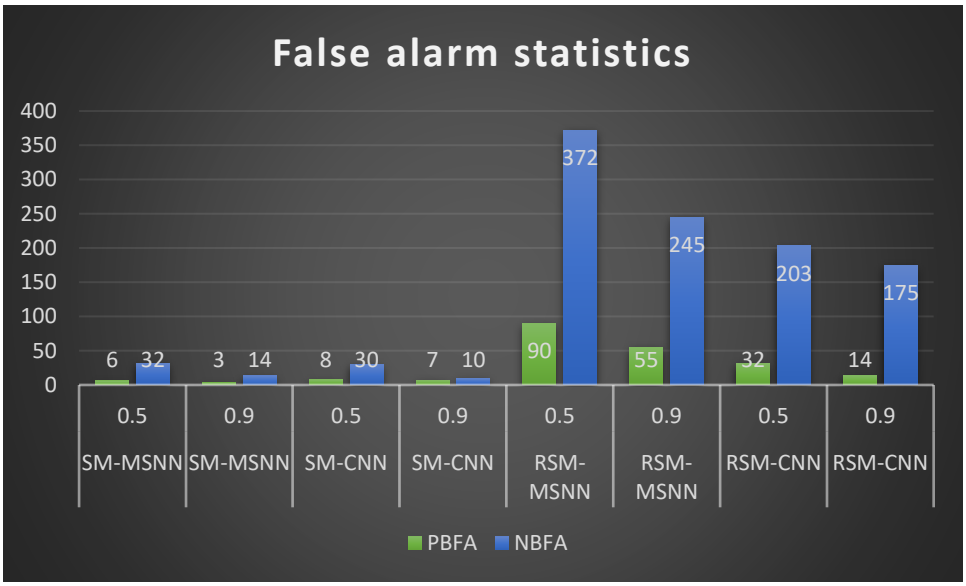


Figure 4.56 Experiment 1: False alarm statistics in test data2 for all algorithms using thresholds 0.5 & 0.9. Positive bag false alarms are in green (Figure 4.48). Negative bag false alarms are in green (Figure 4.49).

In the three tables representing the training dataset (Table 4.3), test data1 (Table 4.3) and test data2 (Table 4.5), the SM-MSNN algorithm performed better than the other algorithms (SM-CNN, RSM-CNN and RSM-MSNN) when MDT and PMDT scores are considered together. However, when only MDT is considered, SM-MSNN performance

falls all the way to the back leaving the other algorithms (RSM-MSNN, SM-CNN and RSM-CNN) in the lead with relative equal performance. The SM algorithms (SM-MSNN and SM-CNN) generate far less false alarms than RSM algorithms (RSM-MSNN and RSM-CNN).

Two approaches to adapt the MSNN to the MIL framework were employed, but the obtained results are still not satisfactory. An alternative option would be to change the way the bags are created.

4.2.2.2 Experiment 2

In this experiment the positive and negative bags were generated in different ways. For the positive bag, we reduced the number of positive images to 12 instead of 34. Some images were very similar to each other; so, these redundant images were eliminated. Instead, each of the selected images—of the size 128 x 128—yielded three bags. The difference between the three bags is in the number of pixels jumped between two consecutive instances. Figure 4.56 shows how the bags are generated. Using an instance of 40 x 40, the first bag will have one instance recorded every four pixels which yields 484 instances. The second bag will have a jump of six pixels which yields 225 instances. The third bag will have a jump of 10 pixels which yields a bag of 81 instances. The three bags will each have a different number of instances. However, this is not a problem in the multiple instance learning framework.

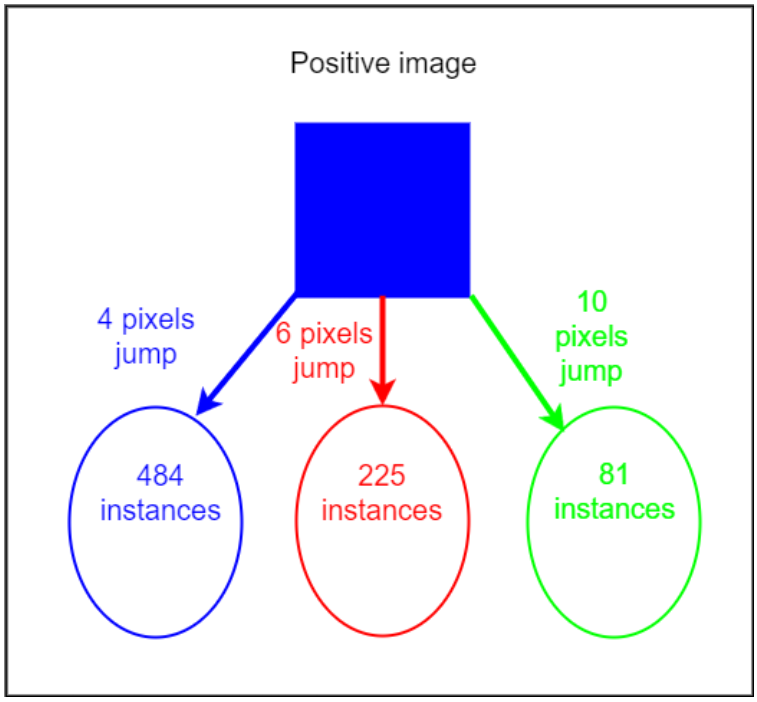


Figure 4.57 Generating multiple bags from one positive image. The blue square represents a positive image. The blue oval is a bag generated using 4 pixels jump. The red oval is a bag generated using 6 pixels jump. The green oval is a bag generated using a 10-pixel jump

For the negative bag, we know that all the instances are negative. A large bag is generated using a small pixel jump. Following the training process, our algorithms were likely to be stuck with one instance, so to learn as much of the background as possible, the bag was split into smaller bags. For example, a negative image (size128 x 128) will produce a bag of 484 instances when a four-pixel jump is used with a target size of 40 x 40. These instances were randomly assigned to four smaller bags of 121 instances each. Figure 4.57 illustrates the process.

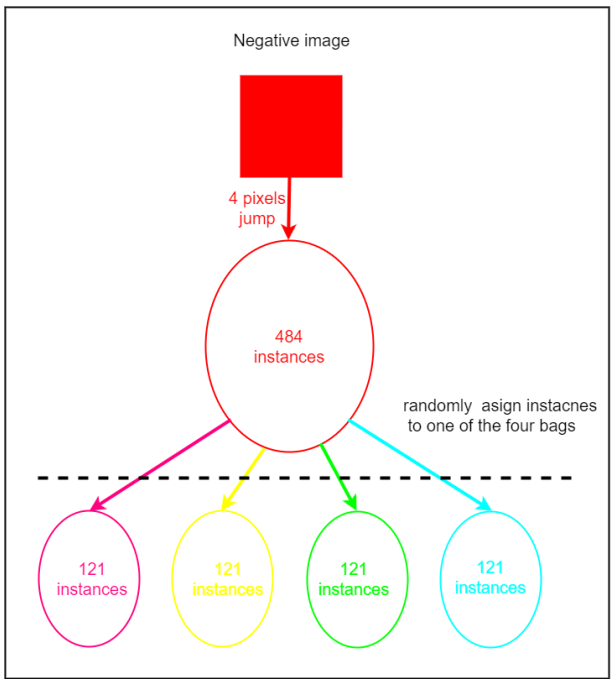


Figure 4.58 Generating multiple negative bags from one negative image. The red square represents a negative image. The red oval is a large bag generated using a four-pixel jump. The smaller magenta, yellow, green and cyan bags were created by splitting the red bag.

The results for Experiment 2 are summarized in the tables below

4.2.2.2.1 Training data: 12 positive images, 43 negative images

The results for testing the algorithm on the training datasets are summarized in table 4.6. The detection scores chart and the false alarms statistics chart are shown in Figure 4.59 and Figure 4.60, respectively. There was an impressive change in the results. Not only did the SM-MSNN take the lead, but for the first time, a perfect instance level detection was achieved, even for the training data. unlike before, all the detections were in MDT. RSM-MSNN and SM-CNN needed the scores of both MDT and PMDT together in order to came close to achieving the same superior instance-level detection as SM-MSNN. RSM-CNN also had a good performance but lower than all the other algorithms

Table 4.6: Experiment 2 Detection results on training data for MIL-MSNN and MIL-CNN with threshold 0.5 and 0.9

	SM-MSNN		SM-CNN		RSM-MSNN		RSM-CNN	
Threshold	0.5	0.9	0.5	0.9	0.5	0.9	0.5	0.9
MDT	12	12	7	7	8	8	3	3
NMDT	0	0	0	0	1	0	2	2
NDT	0	0	2	2	0	1	1	1
PMDT	0	0	3	3	3	3	6	6
PBFA	3/12	3/12	3/12	3/12	26/12	15/12	46/12	42/12
NBFA	0	0	0	0	7/43	5/43	40/43	18/43

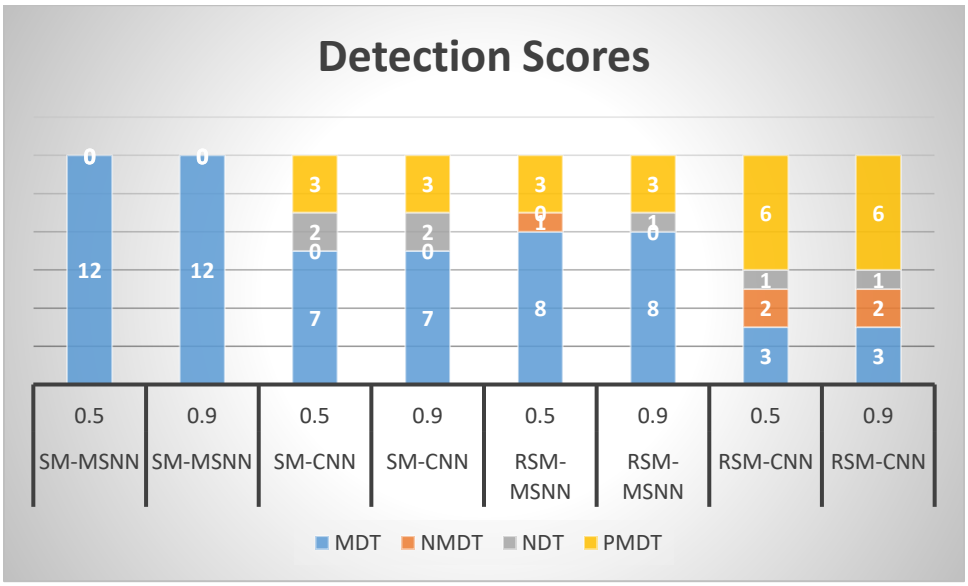


Figure 4.59 Experiment 2 detection scores in training data for all algorithms using thresholds 0.5 & 0.9. Blue is MDT (Figure 4.44). Orange is NMDT (Figure 4.46). Gray is NDT (Figure 4.47). Yellow is PMDT (Figure 4.45)

The SM-MSNN and SM-CNN achieved zero false alarms in the negative images. This is also a first-time achievement with no false alarms in the negative images and only three false alarms in the positive images. Generally, all the algorithms performed well except for RSM-CNN. So, the next step was to check their performance on the test data.

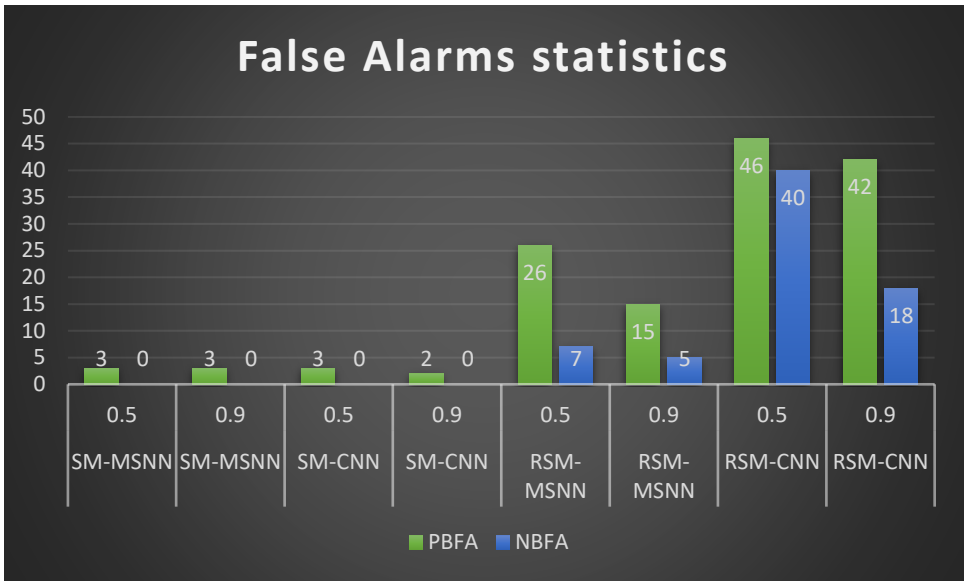


Figure 4.60 Experiment 2 False alarm statistics in training data are shown for all algorithms using thresholds 0.5 & 0.9. Positive bag false alarms are in green (Figure 4.48). Negative bag false alarms are in blue (Figure 4.49).

4.2.2.2.2 Test data1: 32 positive images, 44 negative images

The results for testing the algorithm on test data1 are summarized in Table 4.7. The detection scores chart and the false alarms statistics chart are shown in Figure 4.61 and Figure 4.62, respectively. Again, SM-MSNN achieved a perfect detection on the test data with all images classified in the MDT category. It was followed by the SM-CNN which only came close after both combining MDT and PMDT scores together. Both RSM-MSNN and RSM-CNN lagged far behind in terms of detection. . SM-MSNN and SM-CNN did very well in the false alarm elimination in both positive and negative images. Although SM-CNN had a slight advantage in the negative images, the difference is negligible, especially with SM-MSNN's performance in the detection.

Table 4.7: Experiment 2: Detection results on test data1 for MIL-MSNN and MIL-CNN with threshold 0.5 and 0.9

	SM-MSNN		SM-CNN		RSM-MSNN		RSM-CNN	
Threshold	0.5	0.9	0.5	0.9	0.5	0.9	0.5	0.9
MDT	32	32	26	26	23	23	23	23
NMDT	0	0	1	1	3	3	2	2
NDT	0	0	1	1	1	1	2	2
PMDT	0	0	4	4	5	5	5	5
PBFA	1/32	0	4/32	3/32	17/32	12/32	25/32	17/32
NBFA	6/44	3/44	1/44	0	27/44	13/44	17/44	10/44

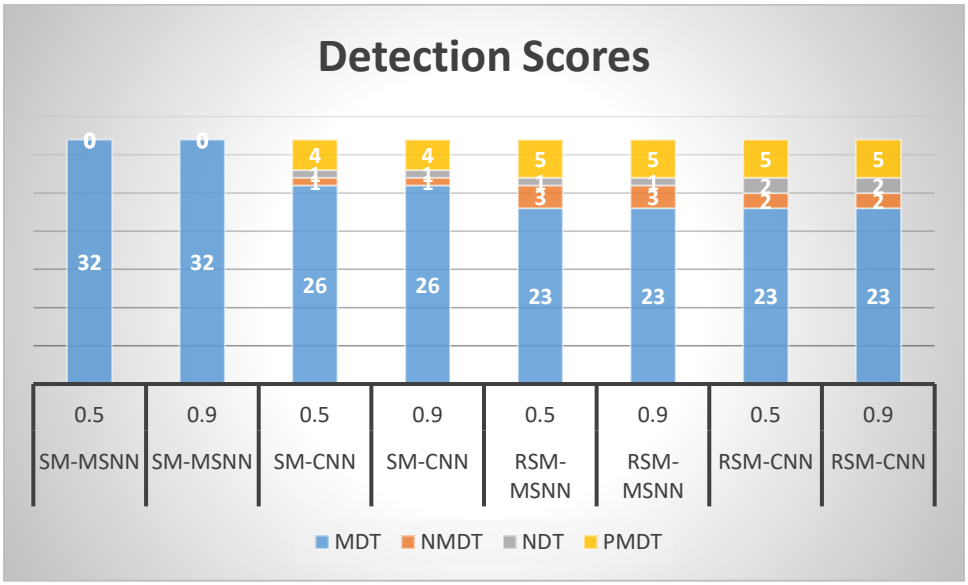


Figure 4.61 Experiment 2 detection scores in test data1 for all algorithms using thresholds 0.5 & 0.9. Blue is MDT (Figure 4.44), Orange is NMDT (Figure 4.46). Gray is NDT (Figure 4.47). Yellow is PMDT (Figure 4.45)

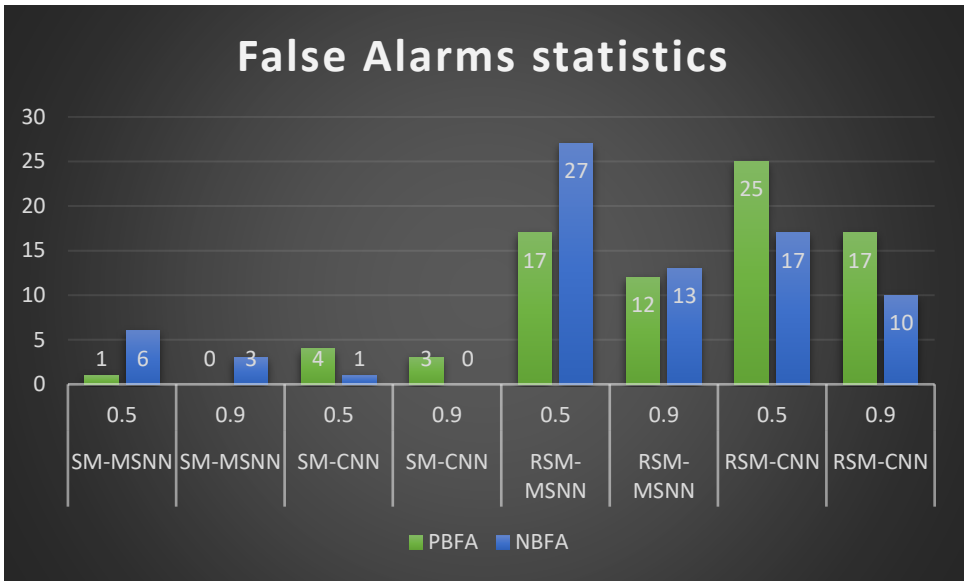


Figure 4.62 Experiment 2 false alarm statistics in test data1 for all algorithms using thresholds 0.5 & 0.9. Positive bag false alarms are in green (Figure 4.48). Negative bag false alarms are in blue (Figure 4.49).

4.2.2.2.3 Test data2: 39 positive images, 88 negative images

The results for testing the algorithm on test data2 are summarized in Table 4.9. The detection scores chart and the false alarms statistics chart are shown in Figure 4.63 and Figure 4.64, respectively. If only the MDT score is considered, then SM-MSMM has a clear advantage over the other algorithms. Whereas if both MDT and PMDT scores are considered, then All the algorithms obtained good detection scores except for RSM-CNN. Both SM-MSNN and SM-CNN continued to have a very good performance in false alarm elimination in both positive and negative images. RSM-MSNN and RSM-CNN had a relatively good performance in eliminating false alarms in the positive images but they still produce high number of false alarms in the negative images.

Table 4.8: Experiment 2 Detection results on test data2 for MIL-MSNN and MIL-CNN with threshold '0.5' and '0.9'

	SM-MSNN		SM-CNN		RSM-MSNN		RSM-CNN	
Threshold	0.5	0.9	0.5	0.9	0.5	0.9	0.5	0.9
MDT	35	35	30	30	30	30	27	27
NMDT	0	0	0	0	2	2	0	0
NDT	4	4	3	3	1	1	5	5
PMDT	0	0	6	6	6	6	7	7
PBFA	6/39	4	3	3	19/39	10/39	14/39	10/39
NBFA	11/88	4/88	0	0	86/88	58/88	117/88	42/88

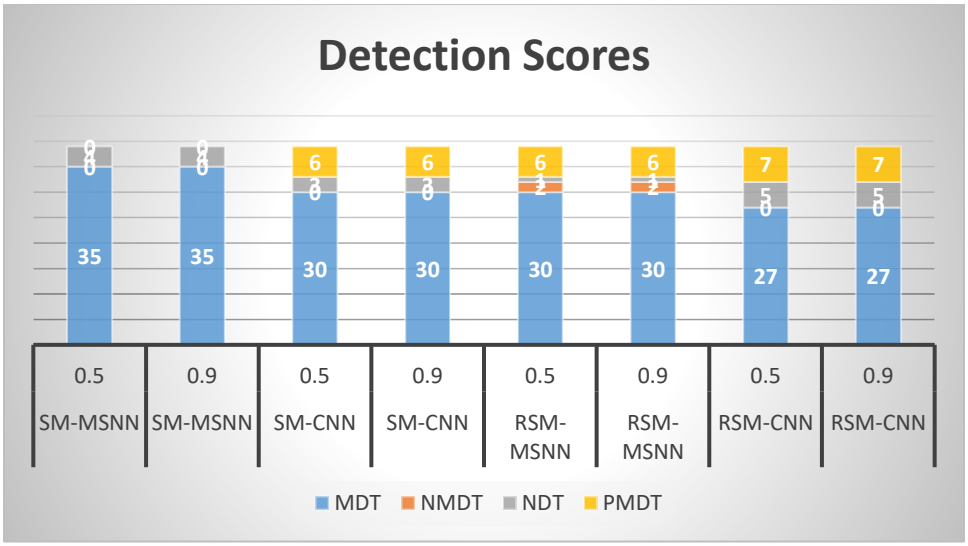


Figure 4.63 Experiment 2: Detection scores in test data2 for all algorithms using thresholds 0.5 & 0.9. Blue is MDT (Figure 4.44)., Orange is NMDT (Figure 4.46). Gray is NDT (Figure 4.47). Yellow is PMDT (Figure 4.45)

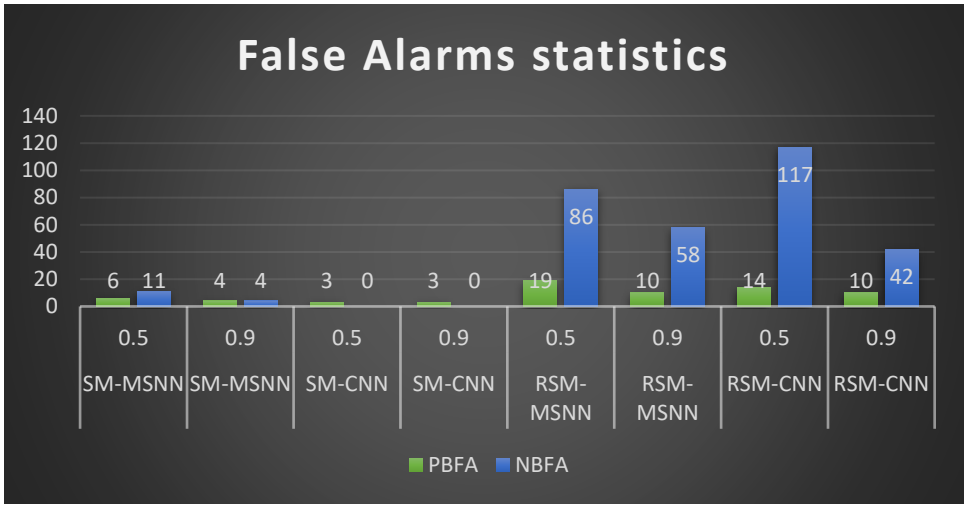


Figure 4.64 Experiment 2 False alarm statistics in test data2 for all algorithms using thresholds 0.5 & 0.9. Positive bag false alarms are in green (Figure 4.48). Negative bag false alarms are in blue (Figure 4.49).

The bag generation method used in this experiment had a twofold objective.1) to help the algorithm distinctively learn the target from the background, especially the border area. This can be seen in the decrease of PMDT scores and increase of MDT scores. The second objective is to increase the chances for the algorithm to learn more background to reduce the number of false alarms which was clearly observed in the four algorithms.

CONCLUSIONS

The multiple instance learning (MIL) framework opens a lot of possibilities for developing computer vision applications, all while reducing the amount of preparation work before training. In our approach, the MSNN approach was used for detecting vehicles in aerial images. The algorithm had a good performance in the supervised learning framework. However, that required exact labeling of the center of the target in each training image.

Adapting our MSNN to the multiple instance learning framework yielded many possibilities. Selecting one of them was not an easy process. Finally, we ended up with two structures, the SM-MSNN and the RSM-MSNN. These approaches were based on the definition of a positive bag, that is, a bag is considered positive if at least one instance is positive. Therefore, we focused on finding and maximizing instances that allowed the bag to be classified correctly. This was possible in SM-MSNN. However, to protect the algorithm from getting stuck with a less optimal instance, RSM-MSNN used a random roulette selection of the top n instances.

Each of the two MSNN algorithms was compared with a similar CNN. The conditions were set for an equal ground comparison. While MSNN used two structuring elements, it only produced one feature map using both. Meanwhile, CNN was trained with two kernels, thereby producing two feature maps. CNN was also allowed to use an activation function in its feature extraction layer, but MSNN did not employ one. Aside from these two conditions, the remaining network parameters were all the same.

The multiple instance learning framework allowed us to perform bag (image) level classification. However, a correct bag classification does not necessarily guarantee a successful instance level classification, i.e., a correct detection of the target. the SM-MSNN had a small edge over the SM-CNN in bag level classification. In instance level classification, we have two important parameters. The MDT which is considered the optimal case where the vehicle is nicely placed in the center of the instance. We also have the PMDT where only a part of the vehicle appears in the instance generating the highest confidence in the image. If we evaluate using MDT only, then MSNN algorithms performance is bad or medium at best. But if we add the PMDT score then SM-MSNN becomes the best among the four algorithms. while this results is good, we observe how SM-MSNN depends heavily on PMDT category.

When the bag generation approach was changed, significant improvement was achieved, especially for SM-MSNN. It achieved the only perfect detection on training data as well as on test data1. Most of the algorithms performed well in test data2 but the lead was still achieved by SM-MSNN followed by the RSM-MSNN.

The MSNN shows a great potential in detecting targets in aerial images using the MIL framework. Especially that no preprocessing of the images was used. Using image patches as instances was enough for the algorithm to the learn the target in the images when the appropriate bag generation approach was used. Bag generation remains the most important factor for any algorithm in order for it to operate within the MIL framework. The MSNN used had one feature extraction layer. While it was effective in detection, a deeper MSNN can be tested to see if it can compensate for the use of a complicated bag generation approach. The MSNN was tested on aerial images. Although aerial images have a lot of

similarity to satellite images, MSNN can be used to detect targets in satellite images. The results obtained from both type images can be compared.

BIBLIOGRAPHY

[1] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov 1998.

[2] D.C. Ciresan, A. Giusti, L.M. Gambardella, and J. Schmidhuber, "Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images," *Proc., NIPS2012*.

[3] A. Krizhevsky, I. Sutskever, G.E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks," *Proc., NIPS2012*.

[4] Y. Won. "Nonlinear Correlation Filter and Morphology Neural Networks for Image Pattern and Automatic Target Recognition," PhD thesis, University of Missouri - Columbia. 1995

[5] F.Y. Shih "Image Processing and Pattern Recognition Fundamentals and Techniques" Institute of Electrical and Electronics Engineering. 2010.

[6] T.G. Dietterich, R.H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles." *Artificial intelligence* 89, no. 1: 31-71. 1997

[7] O Maron, Learning from ambiguity [Dissertation]. Department of Electrical Engineering and Computer Science, MIT, Jun. 1998.

[8] Z-H. Zhou, Multi-Instance Learning: A Survey, National Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China

[9] J. Wang and J-D Zucker. "Solving the multiple-instance problem: a lazy learning approach". In *Proceedings of the 17th International Conference on Machine Learning*, San Francisco, CA, pp.1119–1125, 2000.

[10] Y. Chevaleyre and J-D. Zucker. "Solving multiple-instance and multiple-part learning problems with decision trees and decision rules. Application to the mutagenesis problem". *In Proceedings of the 14th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Ottawa, Canada, LNAI 2056, pp.204–214, 2001.

[11] Zhou, Z.-H. and Zhang, M.-L.: Neural networks for multi-instance learning, Technical Report, AI Lab, Computer Science & Technology Department, Nanjing University, China, Aug. 2002

[12] M.-L. Zhang and Z.-H. Zhou. "Improve multi-instance neural networks through feature selection". *Neural Processing Letters*, vol.19, no.1, pp.1–10, 2004.

[13] M. Nixon and A. Aguado "Feature Extraction and Image Processing" Academic Press. 2008.

[14] B. Babenko, M. H. Yang and S. Belongie, "Visual tracking with online Multiple Instance Learning," *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, pp. 983-990, 2009.

[15] O. Maron, and A. L. Ratan. "Multiple-Instance Learning for Natural Scene Classification." *In ICML*, vol. 98, pp. 341-349. 1998.

[16] P. Viola, J. C. Platt, and C. Zhang "Multiple Instance Boosting for Object Detection" *Proc., NIPS2005*.

[17] Yixin Chen, Jinbo Bi and J. Z. Wang, "MILES: Multiple-Instance Learning via Embedded Instance Selection," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 12, pp. 1931-1947, Dec. 2006.

[18] D. Haun, K. Hummel, and M. Skubic, "Morphological Neural Network Vision Processing for Mobile Robots", University of Missouri-Columbia, 1997.

[19] Yonggwan Won, P. D. Gader and P. C. Coffield, "Morphological shared-weight networks with applications to automatic target recognition," in *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1195-1203, Sep 1997.

[20] B. D. Pawar, and V. T. Humbe "Morphology Based Composite Method for Vehicle Detection from High Resolution Aerial Imagery", *VNSGU Journal of Science and Technology,* vol. 4, no. 1, pp, 50-56, Jul 2015

[21] H. ZHENG, "Automatic Vehicles Detection from High Resolution Satellite Imagery Using Morphological Neural Networks", *Proceedings of the 10th WSEAS International Conference on Computers*, Vouliagmeni, Athens, Greece, pp. 608-613, July 13-15, 2006

[22] Z. Zheng, X. Wang, G. Zhou and L. Jiang, "Vehicle detection based on morphology from highway aerial images," *IEEE International Geoscience and Remote Sensing Symposium*, Munich, pp. 5997-6000, 2012.

[23] L. Cao, C. Wang and J. Li "Vehicle detection from highway satellite images via transfer learning" *Information Sciences* 366, pp. 177–187,2016.

[24] T. Zhou, L. Gu, R. Ren and Q. Cao "The Research of Road and Vehicle Information Extraction Algorithm Based on High Resolution Remote Sensing Image" *Proc. of SPIE* Vol. 9977-99770H-1, 2016.

[25] H. Wei *et al*., "Vehicle detection from parking lot aerial images," *IEEE International Geoscience and Remote Sensing Symposium - IGARSS*, Melbourne, VIC, pp. 4002-4005, 2013.

[26] Mi Shu and Shihong Du, "Geoscene-based vehicle detection from very-high-resolution images," *4th International Workshop on Earth Observation and Remote Sensing Applications (EORSA)*, Guangzhou, pp. 295-299, 2016.

[27] L. Abraham, M. Sasikumar, "Vehicle Detection and Classification from High Resolution Satellite Images", *ISPRS Technical Commission I Symposium*, Denver, Colorado, USA, 17 – 20, Nov 2014.

[28] Z. Hong, H. Xue-min "An antibody networks approach for vehicle detection from high resolution satellite imagery" *Journal of Remote Sensing 13*, 913-927, 2009.

[29] M. A. Khabou and L. F. Solari, "A Morphological Neural Network-Based System for Face Detection and Recognition," *Proceedings of the IEEE SoutheastCon*, Memphis, TN, pp. 296-301, 2006.

[30] H. Sahoolizadeh, M. Rahimi and H. Dehghami, 2008, "Face Recognition using morphological shared weight neural networks" In *Proceedings of World Academy of Science, Engineering and Technology*, vol. 35, pp. 556-559, 2008.

[31] P. K. Podder, D. K. Sarker, and D. Kundu, "Real-Time Face Recognition System Based on Morphological Gradient Features and ANN" *Global Journal of researches in engineering Electrical and electronics engineering Volume 12 Issue 2 Version 1.0,* Feb 2012

[32] R. Lala and A. K. Singh, "MSNN based techniques for Face and Eye Recognition", *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* 2, no. 1, pp-233, (2013).

[33] D. N. Chandrappa, and M. Ravishankar" Gabor Wavelets and Morphological Shared Weighted Neural Network Based Automatic Face Recognition", *Signal & Image Processing: An International Journal (SIPIJ),* Vol.4, No.4, 61-70, Aug 2013.

[34] C. Ke, "White Blood Cell Detection Using a Novel Fuzzy Morphological Shared-Weight Neural Network," *2008 International Symposium on Computer Science and Computational Technology*, Shanghai, pp. 532-535, 2008.

[35] H. Zheng, L. Pan, and L. Li, "A Morphological Neural Network Approach for Vehicle Detection from High Resolution Satellite Imagery," *Proc. Int'l Conf. Neural Information Processing*, 2006.

[36] A. Stuart, I. Tsochantaridis and T. Hofmann. "Support vector machines for multiple-instance learning." *Advances in neural information processing systems*, 577-584, 2003.

[37] B. Babenko, M. H. Yang and S. Belongie, "Visual tracking with online Multiple Instance Learning," *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, pp. 983-990, 2009.

[38] B. Babenko, M. H. Yang and S. Belongie, "Robust Object Tracking with Online Multiple Instance Learning," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1619-1632, Aug. 2011

[39] Z. Qi, and S. A. Goldman. "EM-DD: An improved multiple-instance learning technique." *In NIPS, vol. 1*, pp. 1073-1080, 2001.

[40] D. E. Rumelhart, J. L. McLelland, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Cambridge, MA: MIT Press, 1986.

[41] S. Ali and M. Shah, "Human Action Recognition in Videos Using Kinematic Features and Multiple Instance Learning," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 2, pp. 288-303, Feb. 2010

[42] Z. Qi, S. A. Goldman, W. Yu, and J. E. Fritts. "Content-based image retrieval using multiple-instance learning." *In ICML,* vol. 2, pp. 682-689, 2002.

[43] R. Soumya, and M. Craven. "Supervised versus multiple instance learning: An empirical comparison." *In Proceedings of the 22nd international conference on Machine learning*, pp. 697-704, 2005.

[44] A. Stuart, T. Hofmann, and I. Tsochantaridis. "Multiple instance learning with generalized support vector machines." *In AAAI/IAAI*, pp. 943-944, 2002.

[45] P. Auer. "On learning from multi-instance examples: empirical evaluation of a theoretical approach." *In Proceedings of the 14th International Conference on Machine Learning*, Nashville, TN, pp.21–29, 1997.

[46] A. Blum and A. Kalai. "A note on learning from multiple-instance examples." *Machine Learning*, vol.30, no.1, pp.23–29, 1998.

[47] F. Rosenblatt "The perceptron: A probabilistic model for information storage and organization in the brain." *Psychological Review,* 65, no. 6: 386, 1958.

[48] M. Minsky, and S. Papert. "Perceptrons." 1969.

[49] P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences." PhD thesis, Harvard University, 1974.

[50] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. "1-norm Support Vector Machines." *In NIPS, vol. 15*, pp. 49-56, 2003.

[51] O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. In M.I. Jordan, M.J. Kearns, and S.A. Solla, Eds. Advances in Neural Information Processing Systems 10, Cambridge, MA: MIT Press, pp.570–576, 1998

[52] G. Papandreou, I. Kokkinos and P. A. Savalle, "Modeling local and global deformations in Deep Learning: Epitomic convolution, Multiple Instance Learning, and sliding window detection," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, pp. 390-399, 2015.

[53] M. Sun, T. X. Han, M. C. Liu, and A. Khodayari-Rostamabad. "Multiple Instance Learning Convolutional Neural Networks for Object Recognition." *arXiv* preprint arXiv:1610.03155, 2016.

[54] R. G. Cinbis, J. Verbeek and C. Schmid, "Weakly Supervised Object Localization with Multi-Fold Multiple Instance Learning," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 1, pp. 189-203, Jan. 1 2017.

[55] P. O. Pinheiro and R. Collobert, "From image-level to pixel-level labeling with Convolutional Networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, pp. 1713-1721, 2015.

[56] H. Yang, J. T. Zhou, Y. Zhang, B. B. Gao, J. Wu and J. Cai, "Exploit Bounding Box Annotations for Multi-Label Object Recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, pp. 280-288, 2016.

[57] C. Zitnick and P. Dollar, "Edge boxes: Locating object proposals from edges," *in Proc. Eur. Conf. Comput. Vis*, pp. 391–405., 2014.

[58] X.-S. Wei and Z.-H. Zhou, "An empirical study on image bag generators for multi-instance learning," *Machine Learning*, pp. 1–44, 2016.

[59] C. Yang, M. Dong, and J. Hua, “Region-based Image Annotation using symmetrical Support Vector Machine-based Multiple-Instance Learning,” *in CVPR*, 2006.
[60] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, “Visual categorization with bags of key points,” *in ECCV*, 2004.
[61] H. Lu, Q. Zhou, D. Wang and R. Xiang, "A co-training framework for visual tracking with multiple instance learning," *Face and Gesture 2011*, Santa Barbara, CA, pp. 539-544, 2011.
[62] J. Wang, B. Li, W. Hu and O. Wu, "Horror video scene recognition via Multiple-Instance learning," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Prague, pp. 1325-1328, 2011.

[63] P.M. Long and L. Tan. “PAC learning axis-aligned rectangles with respect to product distributions from multiple-instance examples.” *Machine Learning,* vol.30, no.1, pp.7– 21, 1998.

[64] J. Serra, Image Analysis and Mathematical Morphology, Vol. 2, Academic Press, New York, N.Y ,1988.