الجمهوريّة الجزائريّة الديمقراطيّة الشعبيّة

People's Democratic Republic of Algeria

وزارة التعليم العالي والبحث العلمي

Ministry of High Education and Scientific Research

جامعة غرداية

University of Ghardaia

كليّة العلوم والتكنولوجيا

Faculty of Science and Technology

قسم الرياضيات والإعلام الآلي

Mathematics and Computer Science Department

Registration N°:
/…../…../…../…../…..

Thesis

For obtaining the master's degree

Domain: Artificial Intelligence and Linguistics
Field: Computer Science
Specialty: Intelligent Systems for Knowledge Extraction

Theme

Bio-inspired algorithm for security in Twitter case of image data

Submitted on 30/09/2021

By
Nacer BOUAL & Bahmed DEDJELL

Examined by the jury composed of:

Dr. Nacera BRAHIM	MCB	University of Ghardaia	Examiner
Dr. Houssem Eddine DEGHA	MCB	University of Ghardaia	Examiner
Dr. Bouhani ABDELKADER	MCB	University of Ghardaia	Advisor

Academic Year: 2020/2021

Abstract

In our time, the amount of information and tweets are increasing on Twitter. Unfortunately, we found that Twitter is a popular place for spammers, which share unwanted messages that may contain malicious software, advertisements, or links that contain malicious sites. As a means of avoiding text-based filters, spammers inject spam text onto images, a process known as image spam. so. How can we detect these images and know the unwanted messages from it? What are the possible algorithms to detect it ? This is what we will address in this research. In our thesis, we introduce Some Learning techniques used to classify images as spam or ham and bio-inspired algorithm which used to optimize the problem, at the experimental level we design convolutional neural network architectures using the particle swarm optimization algorithm in order to find the optimal network architecture of convolutional neural networks.

Keywords— Learning techniques, Bio-inspired algorithm, Convolutional neural network, Particle swarm optimization

الملخص

في الوقت الحاضر، تتزايد كمية المعلومات والتغريدات على موقع تويتر. لسوء الحظ، وجدنا أن تويتر مكان شائع لمرسلي البريد العشوائي، الذين يشاركون الرسائل غير المرغوب فيها التي قد تحتوي على برامج أو إعلانات أو روابط ضارة تحتوي على مواقع ضارة. كوسيلة لتجنب عوامل التصفية المستندة إلى النص، يقوم مرسلو البريد العشوائي بإدراج نص البريد العشوائي في الصور، وهي عملية تُعرف باسم البريد العشوائي للصور. وبالتالي. كيف يمكننا الكشف عن هذه الصور ومعرفة الرسائل غير المرغوب فيها منها؟ ما هي الخوارزميات الممكنة لاكتشافه؟ هذا ما سنتناوله في هذا البحث. في أطروحتنا، قدمنا بعض تقنيات التعلم المستخدمة لتصنيف الصور على أنها بريد عشوائي أو لا وخوارزمية مستوحاة من الأحياء والتي تستخدم لتحسين المشكلة. على المستوى التجريبي نقوم بتصميم هياكل الشبكات العصبية التلافيفية باستخدام خوارزمية تحسين سرب الجسيمات من أجل العثور على هندسة الشبكة المثلى للشبكات العصبية التلافيفية.

الكلمات الرئيسية: تقنيات التعلم ، الخوارزمية المستوحاة من الحيوية، الشبكة العصبية التلافيفية، تحسين سرب الجسيمات

Dedication

"*In honor of my family, professors and friends, I'm dedicating my dissertation thesis to them. To my parents, who have always encouraged me and instilled in me the value in being persistent thank you for give me strength to chase my dreams. To My siblings have never left my side and encouraging me in every aspect of the life are very special to me thanks for being next to me and showing great interest in my research. To my teachers for doing everything they can to educate and support us. I'm grateful for you. To Tidiwt thank for the support and encouragement throughout my being with you. To my many friends who have been there for me throughout the process, I also dedicate this dissertation to them. This work is dedicated to my best friend, who has been there for me during my entire university years, and I will always be grateful for everything they have done for me.*"

Bahmed

Dedication

"*My dissertation is dedicated to my family, as well as my university family and many friends. A special feeling of gratitude to my loving parents, whose words of encouragement and push for tenacity ring in my ears. My brothers and sisters whose have never left my side and are extremely dear to me. Thank you to my academic adviser for guiding me through this process, and a particular gratitude to my university best buddy. Thank you very much. My love for you all can never be quantified. God's blessings on you.*"

Nacer

Thanks

First and foremost, we would want to thank God for giving us the strength and the patience to accomplish this thesis. We would like to thank my supervisor, Mr. Bouhani Abdelkader, for his valuable advice and assistance over the course of our work. For their kindness and support. I would also like to thank all of the professors in the Mathematics and Computer Science Departments. Finally, we would really like to express our sincere thanks to everyone who has contributed directly or indirectly to the development of this work.

My gratitude goes to those who have provided emotional support for this work: My family and friends.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Introduction

Social networking sites, including Twitter, are a wide space for spreading news, exchanging opinions and ideas among people in the whole world and communicating with each other, but these features also make this space a fertile place for spammers to send their messages. Or their posts, and freedom of expression is one of the rights guaranteed for everyone to express their opinion using what are called tweets. But unfortunately, these Tweets may exceed certain limits and become inappropriate or threatening. In order to make this universe without threats which may even affect the safety of users, it is necessary to have in place automatic protection mechanisms allowing dangerous users to be found in order to protect them. Several algorithms have been implemented for this purpose, but until now due to the complexity of this algorithmic problem, complete security is far from within reach. Among the most promising avenues are organic algorithms inspired by intelligent behavior in living beings (EHO, PSO, ACO, BSO, etc.) which have proven to be very efficient in solving several complexes. As meta-heuristics is part of artificial intelligence, on one side of the other side, it is impossible to separate machine learning from artificial intelligence, due to the fact that it is a type of artificial intelligence that gives computers the ability to learn without being explicitly programmed. In this topic we propose the use of a bio-inspired algorithms as well as machine learning techniques to solve the security problem in the social network twitter.
This is also what made the administrators of these sites looking for solutions to detect these people and stop their accounts or publications by using machine learning algorithms that analyze the posts or incoming messages to detect them, and with the development of time, spammers resorted to using images to post their messages on it, so. How can we detect these images and know the unwanted messages from it? What are the possible algorithms to detect it ? This is what we will address in this research.
The study is divided into three sections: Before diving into this study, the first chapter covers important terminology and terms such as spam, bio-inspired algorithm, Twitter, Social media, Artificial Neural Networks, and meta-heuristic. In the second chapter, we look at the state of the art in image classification utilizing various learning approaches including convolutional neural networks, support vector machines, and multi-layer perceptrons. The Canny edge detector is a feature that is used in this technique. Then we'll discuss Particle Swarm Optimization, which is a technique for optimizing Convolutional Neural Network Architecture in our case. The third chapter discusses the results of the experiments conducted to build the CNN Model and Tuning CNN Hyperparameters using PSO. Finally, the conclusion summarizes everything we've talked about so far.

Chapter 1

Background

1.1 Introduction

In this chapter we present important definitions and terms that need to be explained and clarified before going into this work.
We first focus on the basic concepts as Spam and approach that used in this work as Bio-inspired algorithm and Artificial Neural Networks.Also many Learning Techniques. In addition, our feature The Canny edge detector is a well-known tool for extracting information from images' edges.

1.2 Spam definition

Spamming consists of sending unsolicited messages to a large number of users in an arbitrary manner. Spam has a wide variety of uses, ranging from advertising to online deception and other fraudulent activities. Since sending Spam messages via e-mail has little or no cost, e-mail Spam can be economically viable. In this research, we discuss spam in general, image spam in particular, and we consider related work.

1.3 Bio-inspired algorithm

These are algorithms inspired by nature, after studying the behavior of living organisms in general and in particular the behavior of animals, we develop these algorithms, with the aim of improving results or finding solutions to difficult problems or not solved by traditional algorithms.
There are more than 200 algorithms inspired by nature.[11]
That can be divided existing algorithms into six major categories illustrated in (Figure 1.1 Classification using the inspiration source based taxonomy) :

- Swarm Intelligence (SI) based : It is the collective behavior of dispersed, self-organized systems in natural and artificial contexts. The term was coined in the context of robotic systems, but it has now come to mean the formation of collective intelligence from a number of basic agents who follow simple behavioral norms.

- Breeding-based Evolutionary Algorithms : This category includes population-based algorithms based on Natural Evolution concepts. Each person in the population represents a solution to the problem and has a fitness value associated with it (namely, the value of the problem objective function for that solution).

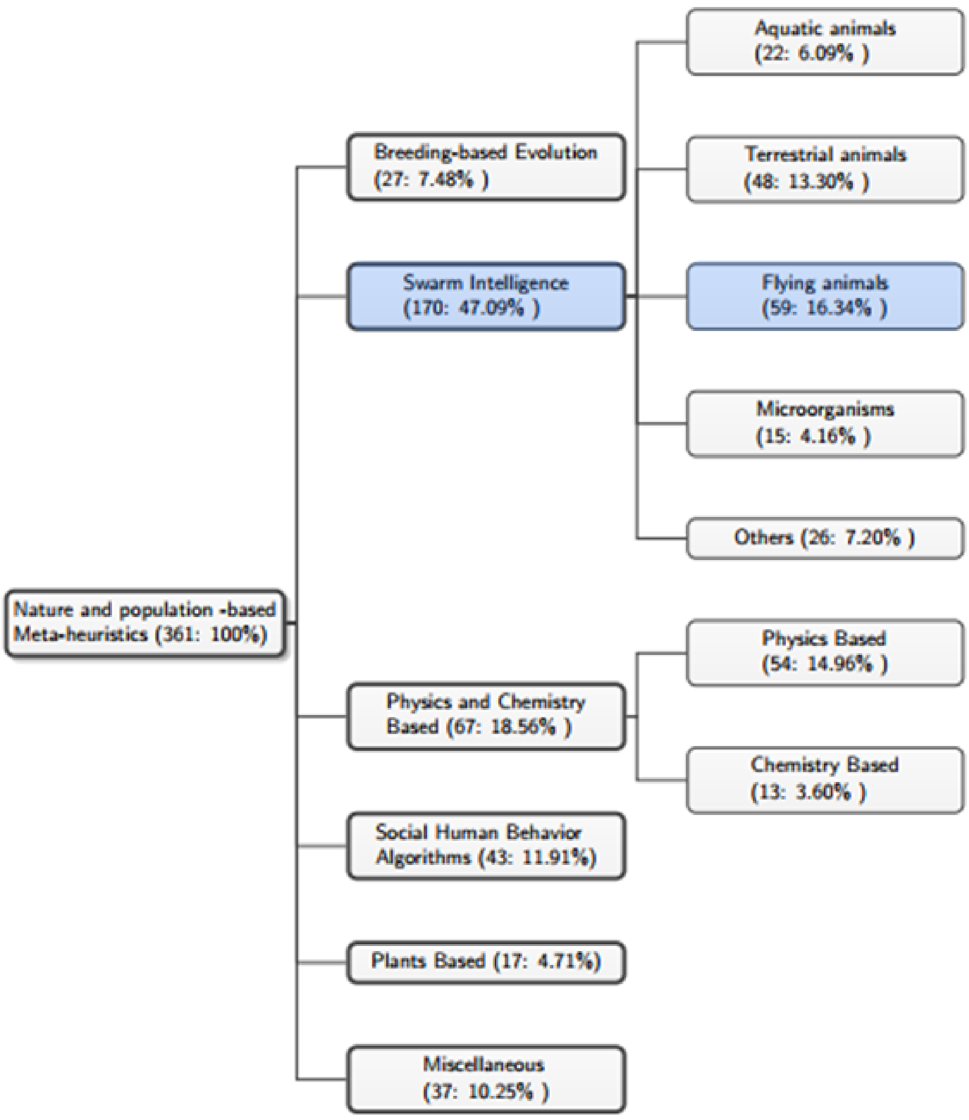
- Physics/Chemistry based Algorithms : Algorithms under this category are characterized by the fact that they imitate the behavior of physical or chemical phenomena, such as gravitational forces, electromagnetism, electric charges and water movement (in relation with physics-based approaches), and chemical reactions and gases particles movement as for chemistry-based optimization algorithms.

- Social Human Behavior based Algorithms : Algorithms falling in this category are inspired by human social concepts, such as decision making and ideas related to the expansion/competition of ideologies inside the society as ideology , or political concepts such as the Adolescent Identity Search Algorithm.

- Plants based Algorithms : This category encompasses all optimization methods that are inspired by plants in their search process.

- Miscellaneous Sources of Inspiration : The algorithms that do not fit into any of the previous categories are included in this category. Although this defined category is heterogeneous and does not exhibit any uniformity among the algorithms it represents, its inclusion in the taxonomy serves as an exemplifying fact of the many different sources of inspiration that exist in the literature.[1]

The strength of these algorithms lies in the fact that they provide solutions close to the ideal and at the same time in a time acceptable and with available tools(hardware).



Nature and population -based Meta-heuristics (361: 100%)
Breeding-based Evolution (27: 7.48%)
Swarm Intelligence (170: 47.09%)
Physics and Chemistry Based (67: 18.56%)
Social Human Behavior Algorithms (43: 11.91%)
Plants Based (17: 4.71%)
Miscellaneous (37: 10.25%)
Aquatic animals (22: 6.09%)
Terrestrial animals (48: 13.30%)
Flying animals (59: 16.34%)
Microorganisms (15: 4.16%)
Others (26: 7.20%)
Physics Based (54: 14.96%)
Chemistry Based (13: 3.60%)

Figure 1.1: Classification using the inspiration source based taxonomy.[1]

1.4 Twitter site

Is a widely used microblogging site where users look for timely and social information such as breaking news, celebrity posts, and hot topics. Users post tweets, which are short text messages limited to 140 characters in length and visible to the user's followers. A follower is someone who chooses to have other people's tweets appear on their timeline. Twitter has been utilized in numerous brand promotions, elections, and as a news medium, and it has been used as a medium for real-time information distribution. Since its inception in 2006, the number of people who utilize it has skyrocketed. Every day, around 200 million tweets are generated as of June 2011. When a new topic becomes popular on Twitter, it is labeled as a trending topic, which can be expressed in short sentences (for example, Michael Jackson) or hashtags (e.g., election). What the Trend2 maintains a list of hot topics on Twitter that is updated on a regular basis. It's fascinating to learn about current events and what interests individuals in other parts of the world. However, hashtags, a person's name, or words in foreign languages make up a large amount of hot topics, and it's often impossible to figure out what they're about. It is consequently critical to group these issues into broad categories in order to facilitate topic comprehension and information retrieval.[12]

1.5 Social Media

It is a site that includes the accounts of people, organizations and associations...., interconnected and interconnected with each other in the form of a spider web, each account presents its ideas and opinions, and it is also considered a space for publishing news and exchanging information and adding to it the discussion of current and important topics, and these sites are very popular It also took an important share of other people's times that reached the point of addiction, so it is considered a powerful tool for spreading awareness or influencing public opinion in both its positive and negative aspects. It has contributed to the overthrow of entire regimes and states, as well as to the establishment of other states, so it is more than just a site, but rather a lethal weapon not to be underestimated.

1.6 Artificial Neural Networks

Artificial Neural Networks have become popular over the last ten years for diverse applications from financial prediction to machine vision. Although these networks were originally proposed as simplified models of biological neural networks, we are concerned here with their application to supervised learning problems.

1.7 Meta-Heuristic definition

In computer science, Artificial Intelligence, and Mathematical optimization, a Heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.

1.8 Different Learning Techniques

1.8.1 Convolutional Neural Networks (CNN):

Convolutional Neural Networks (CNN) are one of the most common types of neural networks (Figure 1.2 : Neural network) used to recognize and classify pictures. CNNs are widely employed in domains such as object detection, face recognition, and so on. In CNN, we use an image as an input, which is represented as an array of pixels (). It will see h x w x d (h = Height, w = Width, d = Dimension) based on the image resolution. Dimension equals 3 for RGB image matrix (3 refers to RGB values red, green, blue), and 1 for grayscale image matrix. [2]

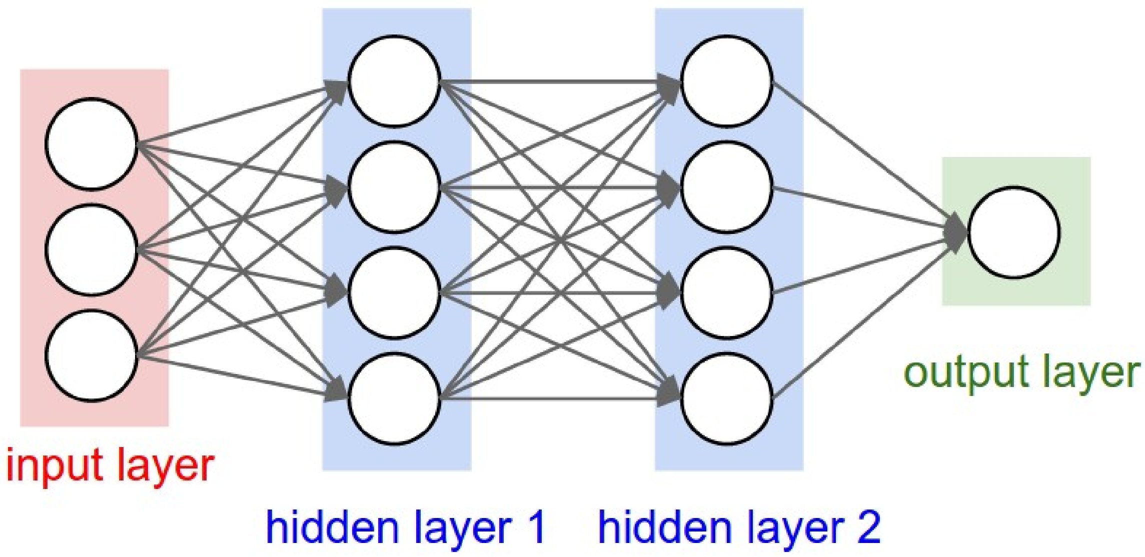


Figure 1.2: Noural Network[2].

- Architecture :

Each input image will be processed using a sequence of Convolution Layers with filters (Kernals), Pooling, and Fully linked layers (FC) So typically has three layers: a Convolutional Layer, Pooling Layer, and Fully Connected Layer as illustrated in (Figure 1.3 : CNN Architecture)

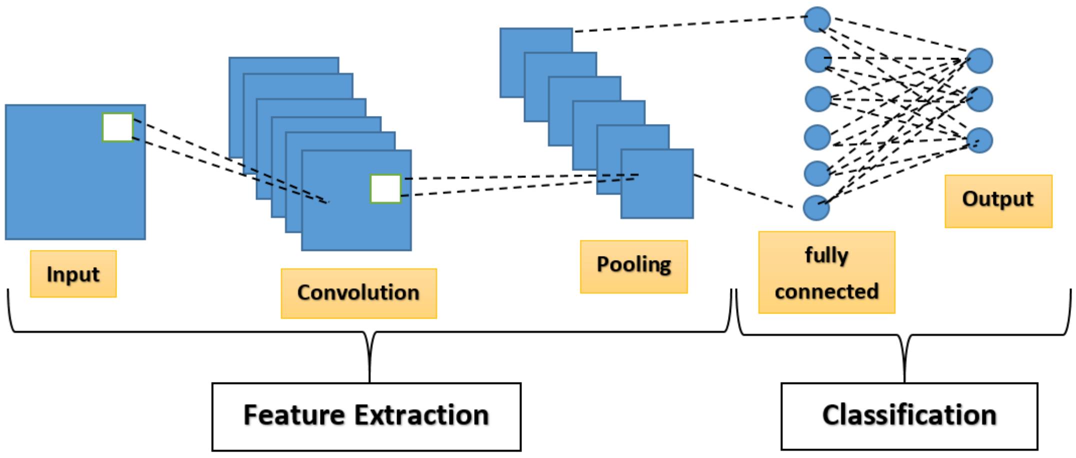


Figure 1.3: CNN Architecture.

Convolution Layer :

C onvolution is the first layer, and its main goal is to extract features like edges, colors, and corners from the input. As we delve deeper into the network, it begins to recognize more complex features such as shapes, digits, face parts, and text. It's a mathematical procedure that requires two inputs : a set of trainable parameters (known as the Filter/Kernel) and a restricted section of the image (known as the restricted portion). - An image matrix(volume) of dimension (h*w*d) - A filter (fh * fw * d) - Outputs a volume dimension (h- fh +1) *(w- fw +1) *1

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**

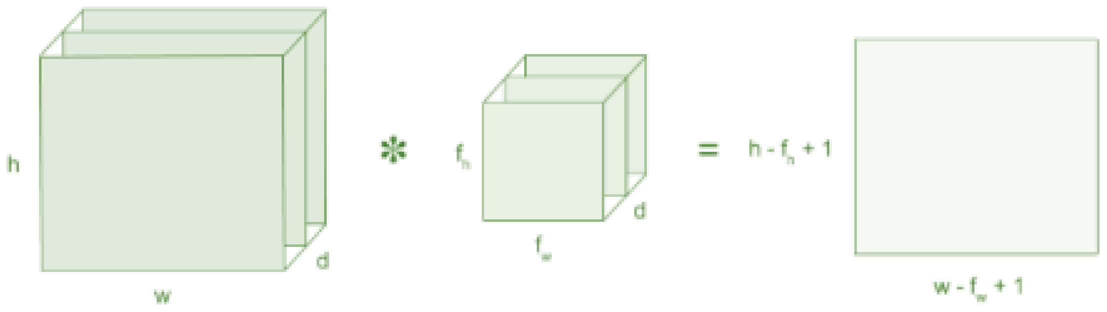


Figure 1.4: Image Matrix Multiplies Kernel or Filter Matrix[2].

C onsider a 3x3 Kernel is convoluted over a 5x5 source image which called "Feature Map" as output shown in (figure1.5 : output matrix).[2]

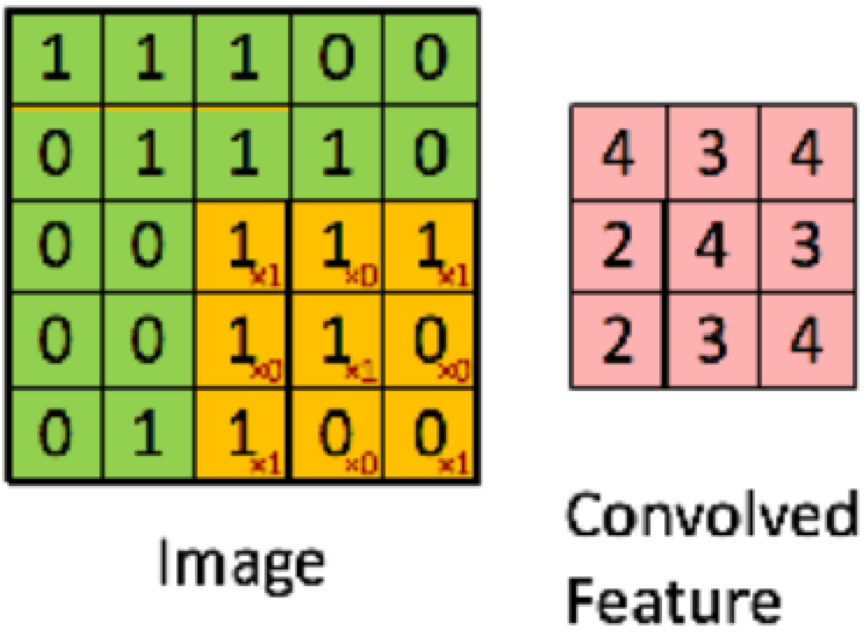


Figure 1.5: 3 x 3 Output matrix[2].

T he output volume is controlled by three hyper parameters: depth, stride, and zero-padding:

- Depth: It's the number of filters we'd like to employ.
- Stride: number of pixels shifts over the input matrix When the stride is set to 1, the filters are moved one pixel at a time. When the stride is two, the filters jump two pixels at a time, and so on.
- Zero-Padding: Filters do not always precisely fit the input image, so we pad it with zeros around the border of the input volume which allow us to control the output volumes' spatial size. [2]

Pooling Layer :

The Pooling layer is responsible for reducing the Convolved Feature's spatial size. This is done to reduce the amount of parameters and computations in the network, as well as to control overfitting by reducing dimensionality. Pooling can be divided into two types: Max Pooling and Average Pooling:

- Max Pooling: is a pooling procedure that return the maximum value from the feature map region covered by the filter like (Figure 1.6 : Max Pooling) After the max-pooling layer, the output would be a feature map with the most important features from the preceding feature map. Also can also be used to reduce noise. It removes all noisy activations and conducts de-noising and dimensionality reduction at the same time.

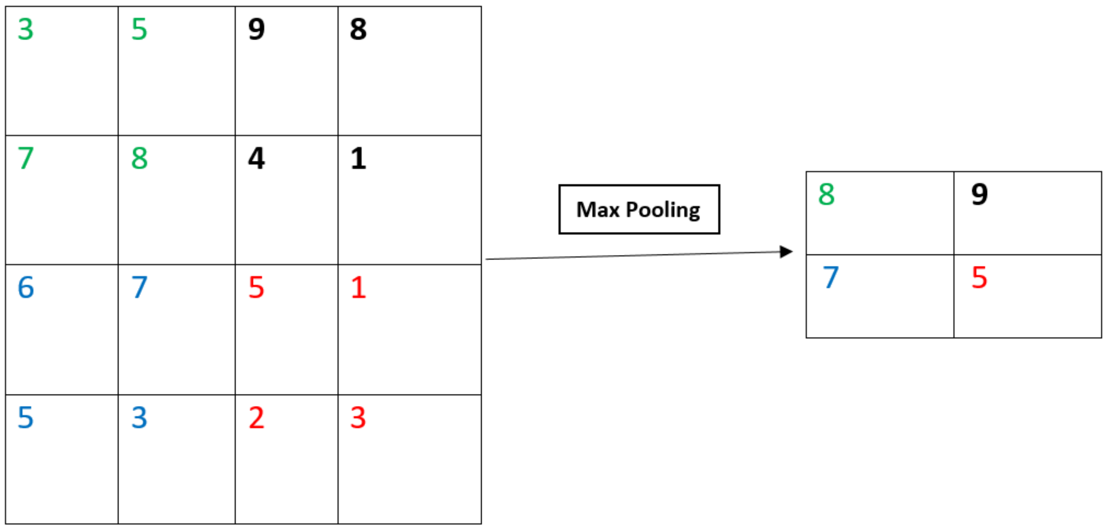


Figure 1.6: Max Pooling.

- Average Pooling: is a pooling procedure that return the average value from the feature map region covered by the filter like (Figure 1.7 : Average Pooling). [13]

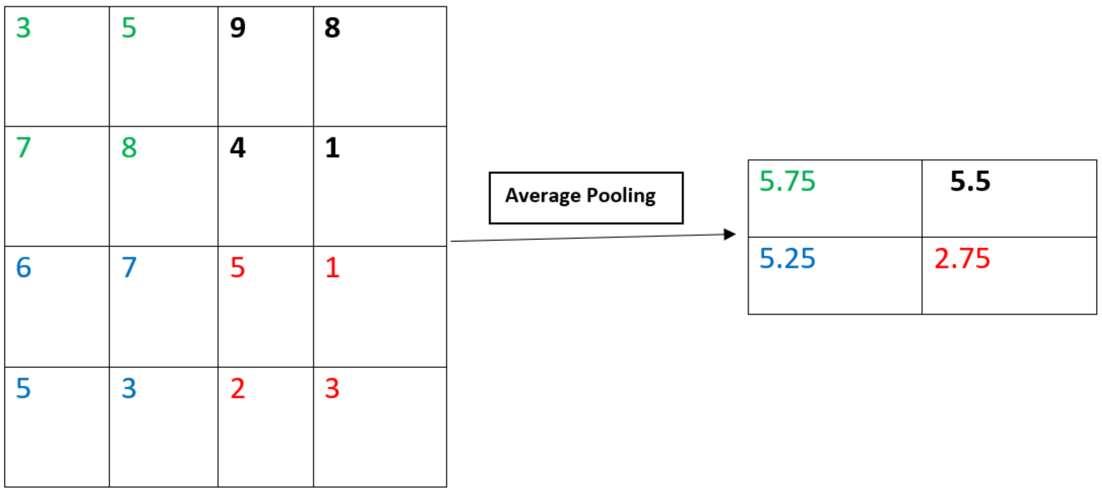


Figure 1.7: Average Pooling

As a result of the Pooling Layer, we now have a matrix comprising the image's main features, and this matrix has even smaller dimensions, which will aid us much in the next phase.[14] The i-th layer of a Convolutional Neural Network is made up of the Convolutional Layer and the Pooling Layer. Depending on the image complexity, the number of such layers might be raised even more to capture low-level features, but at the cost of more computational power. After going through the previous method mean that we have successfully enabled the model to understand the features.[14]

Fully connected layer :

A completely linked layer is the last layer in CNN. We link all of the nodes from the previous layer to this completely connected layer, which conducts the classification operation based on the features retrieved from the previous layers and their various filters (Figure 1.8 : CNN Architecture). [15] This fully connected layer examines the output of the preceding layer and identifies which features are most closely related to a specific class. Finally, we can classify the outputs using an activation function like softmax or sigmoid. [15]

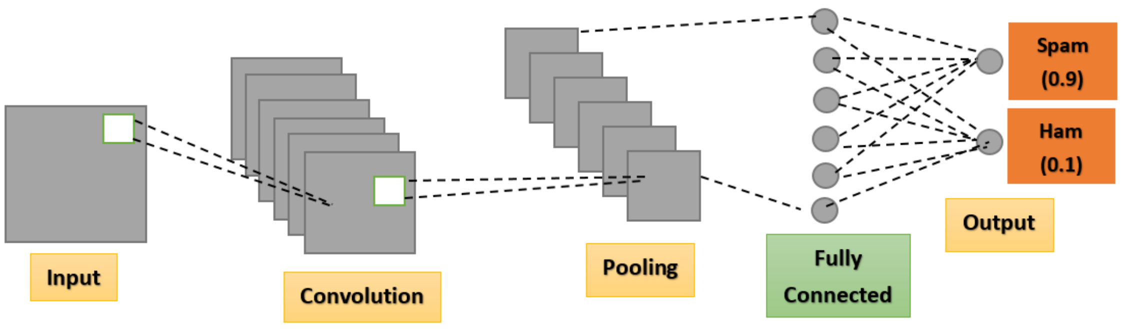


Figure 1.8: CNN Architecture.

1.8.2 Support Vector Machine (SVM):

SVM stands for Support Vector Machine and is a Supervised Machine Learning technique that can be used for classification and regression. SVMs are more typically utilized in classification problems, therefore we'll concentrate on that in this section. represent each data item as a point in n-dimensional space (where n is the number of features), with the value of each feature being the value of a certain coordinate in this algorithm. Then we perform classification by determining the best hyper-plane for separating the two classes.[16]
We use the SVM technique to find the points from both classes that are closest to the line. Support vectors are the names given to these points. The distance between the line and the Support Vectors is now computed. The Margin is the term used to describe the amount of space between two objects. Our goal is to increase the profit margin. The Hyperplane with the greatest margin is the best Hyperplane as illustrated in (Figure 1.9 : Optimal Hyperplane using the SVM algorithm). [17]

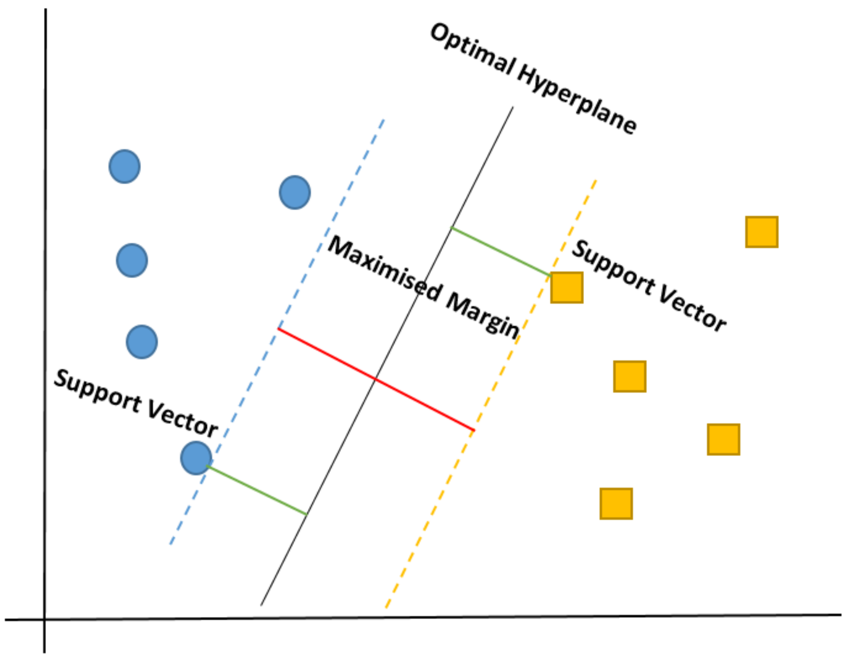


Figure 1.9: Optimal Hyperplane using the SVM algorithm.

- Separating Hyperplane : Hyperplanes are decision-making boundaries that aid in data classification. Different classes can be assigned to data points on either side of the Hyperplane. The Hyperplane's dimension is also determined by the number of features. In the case of one-dimensional data, a Hyperplane will be a point, a line in the case of two-dimensional data, a plane in the case of three-dimensional data, and so on. [3]

    – Not Linearly Seperable: There is no guarantee that the data in the classes will be separable in a linearly. as shown in (Figure 1.10 : Non-linearly separable data)

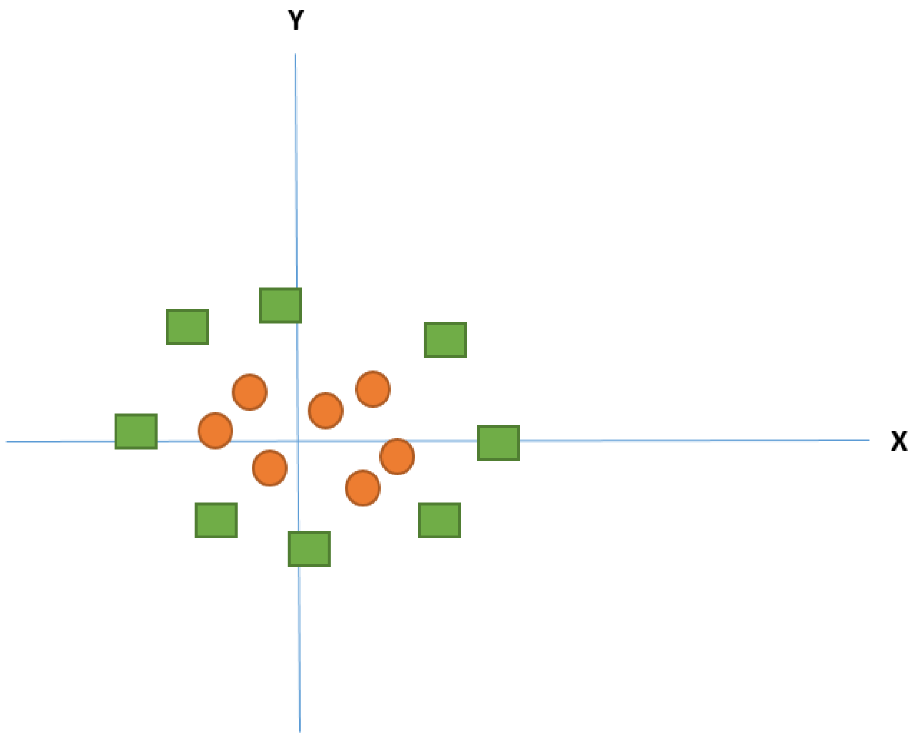


Figure 1.10: Non-linearly Separable Data

there is no line that can be drawn between the two classes. This is a problem that SVM can tackle. We can plot the z feature, which is defined as z= x2 + y2. Using this transformation, we may project this linear separator in higher dimensions back into its original dimensions. this line maps to a circular boundary, as seen in (Figure 1.11 : Dataset on higher dimension).These transformations are called kernels. [17]

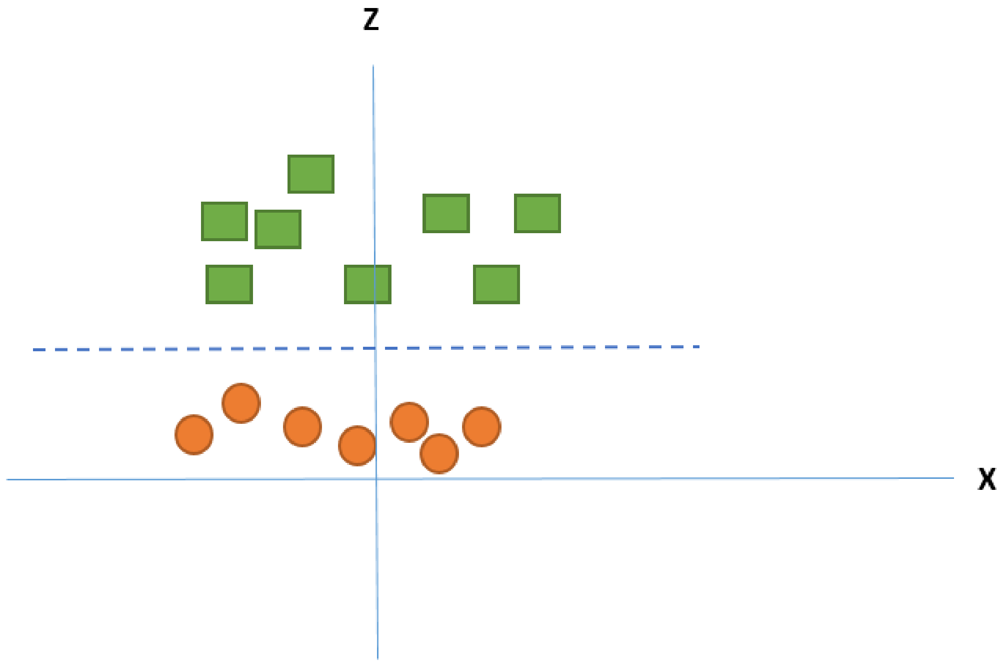


Figure 1.11: Dataset on Higher Dimension.

- Kernel Trick: a way of solving a non-linear problem with a linear classifier. It requires converting linearly inseparable data into linearly separable data such as those shown in (Figure 1.12 : Hyperplanes in 2D and 3D feature space).

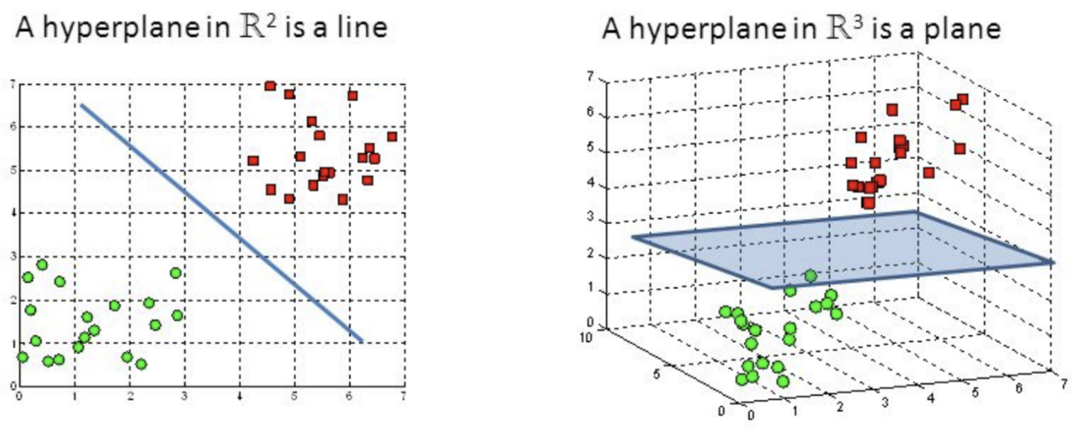


Figure 1.12: Hyperplanes in 2D and 3D Feature Space [3].

The kernel function is used on each data point to translate the non-linear observations into a higher-dimensional space where they can be separated. [18]

We can identify a decision surface that clearly differentiates between distinct classes if we find a way to transfer the data from 2-dimensional space to 3-dimensional space, as shown in (Figure 1.13 : Kernel Trick).

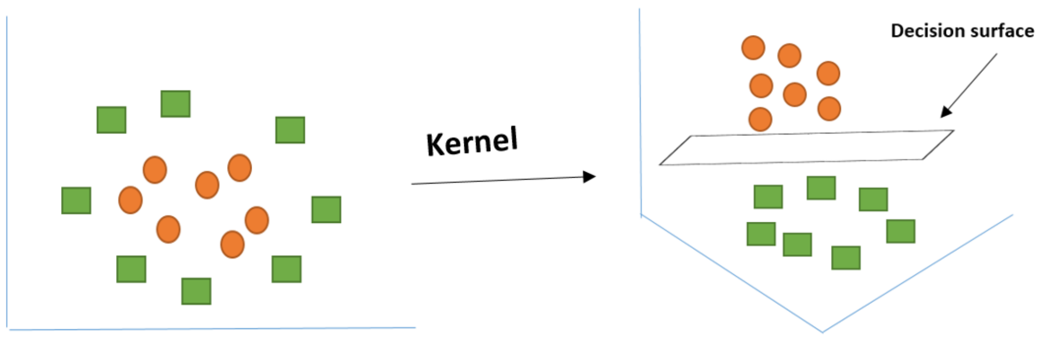


Figure 1.13: Kernel Trick.

The first thing that comes to mind when thinking about this data transformation process is to transfer all of the data points to a higher dimension (in this example, 3-dimensional), determine the boundary, and classify the data. That appears to be adequate. However, as the number of dimensions increases, computations within that space grow more expensive. This is when the kernel technique comes into play. It enables us to work in the original feature space without having to compute the data's coordinates in a higher dimensional space. [19]

Mathematical definition: K (x, y) = <f(x), f(y)>. Here K is the kernel function, x, y are n dimensional inputs. f is a map from n-dimension to m-dimension space. < x, y> denotes the dot product. usually m is much larger than n, calculating typically needs us to first compute f(x), f(y), and then the dot product. Because these two compute stages require manipulations in m-dimensional space, where m can be a big number, they can be quite costly.[18] Simple Example: x = (x1, x2, x3); y = (y1, y2, y3). Then for the function f(x) = (x1x1, x1x2, x1x3, x2x1, x2x2, x2x3, x3x1, x3x2, x3x3) the kernel is K (x, y) = (<x, y>) ^{2}. To make this more intuitive, let's put some numbers in: suppose x = (1, 3, 4); y = (4, 7, 8). Then: f(x) = (1, 3, 4, 3, 9, 12, 4, 12, 16) f(y) = (16, 28, 32, 28, 49, 56, 32, 56, 64) <f(x), f(y)> = 16 + 84 +128 + 84 + 441+ 672 + 128 + 672 + 1024 = 3249 Because f is a mapping from 3-dimensional to 9-dimensional space, there is a lot of algebra involved.[18] Let's try using the kernel instead (Figure 1.14 : kernel function): $K(x, y) = (4 + 21 + 32)^2 = 57$š = 3249 Same result, but much easier to calculate.[18]

$$\begin{aligned} k(\mathbf{x},\mathbf{y}) &= (\mathbf{x}^T\mathbf{y})^2 \\ &= (x_1y_1 + x_2y_2 + x_3y_3)^2 \\ &= \sum_{i,j=1}^{3} x_ix_jy_iy_j \end{aligned}$$

Figure 1.14: Kernel Function.

- Different kernel Functions: Different types of kernel functions are used in SVM algorithms. These functions include Linear, Nonlinear, Polynomial, Radial Basis Function (RBF), and Sigmoid Functions.in this part we will mention both Linear and RBF Kernels :
  - Linear kernel: When the data can be divided using a single line, the Linear Kernel is utilized. One of the most widely utilized kernels is this one. It's typically employed when a data set has a lot of features. Text Classification is an example of a feature with a large number of options, as each alphabet has its own set of options.[20] A Linear Kernel is faster than any other Kernel for training an SVM. Only the (C) Regularisation Parameter must be optimized when training an SVM with a Linear Kernel. When training with various kernels, however, the (Γ) parameter must be optimized, which implies that executing a grid search will normally take longer. Linear Kernel Formula F (x, xj) = sum(x.xj) Here, x, xj represents the data you're trying to classify.[20]
  - RBF kernel: In svm, it is one of the most popular and often utilized kernel functions. It's typically used with non-linear data. When there is no prior knowledge of data, it aids in proper separation. RBF Kernel Formula(Figure 1.15 : RBF kernel)[4].

$$k(\mathbf{x},\mathbf{y}) = e^{-\gamma\|\mathbf{x}-\mathbf{y}\|^2}, \gamma > 0$$

Figure 1.15: RBF kernel [4]

- TUNNING PARAMETERS OF SVM :
  - Regularisation Parameter (C): to regulate the trade-off between decision boundary and correctly classifying training points in order to avoid misclassifying each training example, You'll obtain more training points correctly if c is large as shown in (Figure 1.16: Regularization Parameter).[19]

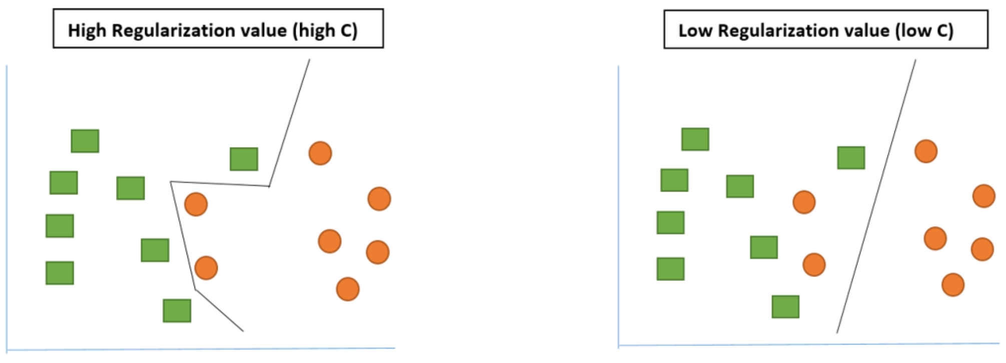

High Regularization value (high C)
Low Regularization value (low C)

Figure 1.16: Regularization Parameter

– Gamma(Γ) : Determines how far a single training example has an impact. This indicates that high Gamma will only evaluate points near the plausible hyperplane, whereas low Gamma will consider sites further away as shown in (Figure 1.17 : Gamma Parameter). [19]

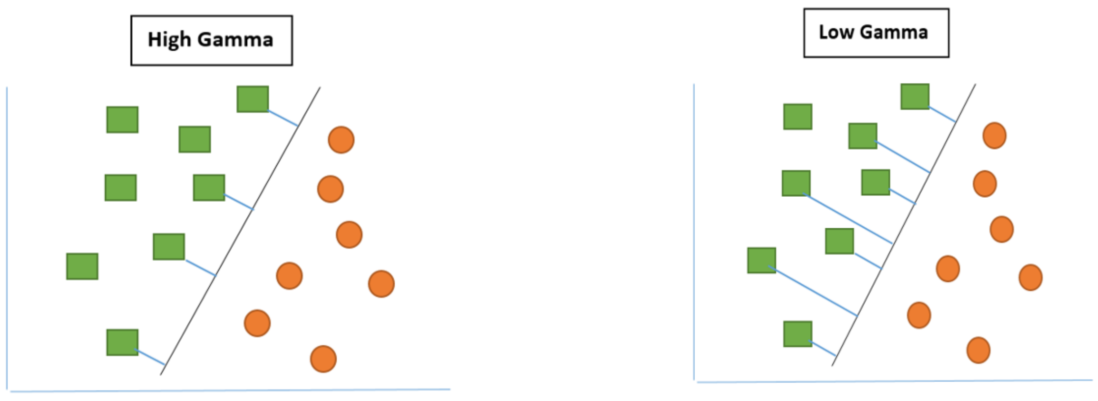


Figure 1.17: Gamma Parameter

- Choosing Linear kernel vs Nonlinear kernel: Andrew Ng gives a nice rule of thumb explanation in [21] starting 14 :46 Key Points : - Use Linear Kernel when number of Features is larger than number of Training example. - Use Gaussian kernel when number of Training example is larger than number of Features. - If number of Training example is larger than 50,000 speed could be an issue when using Gaussian kernel ; hence, one might want to use linear kernel.[21]

1.8.3 Multi-Layer Perceptrons (MLP)

Multi-Layer Perceptrons (MLP) is a subfield of Artificial Neural Network, it has the same simple structure, that has an input layer and an output layer and include one or more Hidden Layers between the input and the output An example of an MLP with two Hidden Layers is given in figure 1.18. Although a perceptrons conceptually simple,the result with different data set in figures suivant :

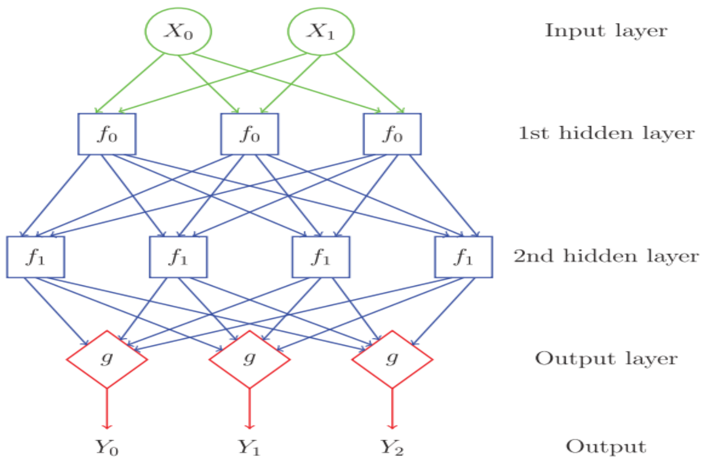


Figure 1.18: MLP With Two Hidden Layers.[5]

1.9 Canny Edge Detector Feature :

Is an edge detection operator that extracts information about edges in photos using a multi-stage approach. It was created in 1986 by John F. Canny. An image with its edges highlighted is called a Canny Image. [7] There are five steps in the Canny Edge Detection algorithm : - Noise Reduction. - Gradient Calculation. - Non-Maximum Suppression. - Double Threshold. - Edge Tracking by Hysteresis. Grayscale images are used in the algorithm. You must first convert the image to grayscale before proceeding with the above steps.[7]

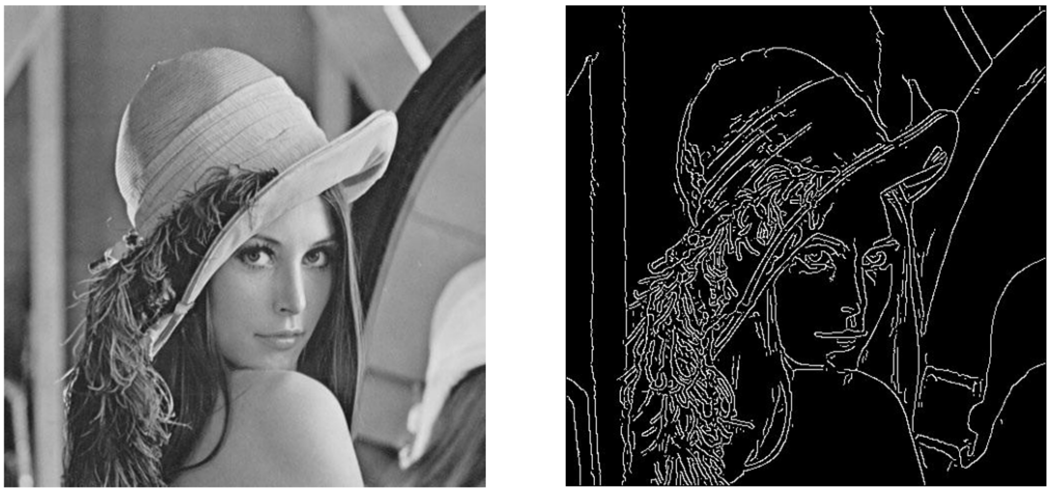


Figure 1.19: Original Image— Processed Image [6]

1.9.1 Noise Reduction:

On the image, apply a Gaussian blur. Before further processing the image, the blur removes some of the noise. To accomplish so, a Gaussian Kernel (3x3, 5x5, 7x7, etc...) is used in conjunction with picture convolution. The size of the kernel is determined by the blurring effect desired. In general, the smaller the kernel, the less noticeable the blur. The following (Figure 1.20: Gaussian Filter kernel equation) is the equation for a Gaussian Filter Kernel of size (2k+1) (2k+1):

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

Figure 1.20: Gaussian Filter Kernel Equation [7]

The standard deviation (sigma σ) is a measure of dispersion. The term "dispersement" describes how far your info is dispersed. It demonstrates how widely your data is dispersed around the mean or average. A low standard deviation implies that the values are close to the set's mean, whereas a high standard deviation suggests that the values are dispersed over a larger range.[22] We get the following result (Figure 1.21 : Noise Reduction with a Gaussian filter) after applying the Gaussian blur.

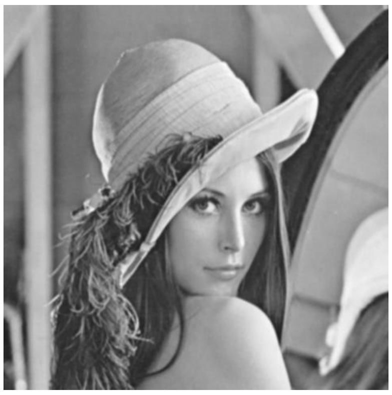


Figure 1.21: Noise Reduction with a Gaussian Filter[6]

1.9.2 Gradient Calculation:

The Gradient computation stage calculates the gradient of the image using edge detection operators to determine the edge strength and direction. Edges represent a change in the intensity of pixels. Applying filters that highlight this intensity change in both directions is the simplest technique to detect it: vertical (y) and horizontal (x). [7] The gradients can be determined by using a Sobel Filter (Figure 1.22 : Sobel Filters).

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Figure 1.22: Sobel Filters [7]

An edge occurs when the color of an image changes, hence the intensity of the pixel changes as well. A Sobel Filter can be used to detect gradients. When the color of an image changes, the pixel's intensity changes as well, so the edge happen.
The magnitude and angle of the directional gradients should then be calculated (Figure 1.23 : Gradient intensity and Edge direction) :

$$|G| = \sqrt{{G_x}^2 + {G_y}^2}$$

$$\angle G = arctan(G_y/G_x)$$

Figure 1.23: Gradient Intensity and Edge Direction [6]

The following (Figure 1.24 : Gradient Intensity) is the output of the image:

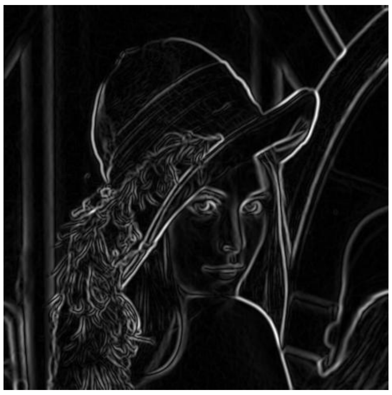


Figure 1.24: Gradient Intensity[6]

Although the outcome is nearly as expected, we can see that some of the edges are dense while others are thin. We'll use the Non-Max Suppression phase to help us deal with the dense ones.[7]

1.9.3 Non-Maximum Suppression:

The final image's edges should be thin. As a result, to thin out the edges, we must use Non-Maximum Suppression. The concept is straightforward: the algorithm iterates through all of the points on the gradient intensity matrix, looking for the pixels with the highest value in the edge directions. [7]
Let's look at an example (Figure 1.25: Example the edge directions).

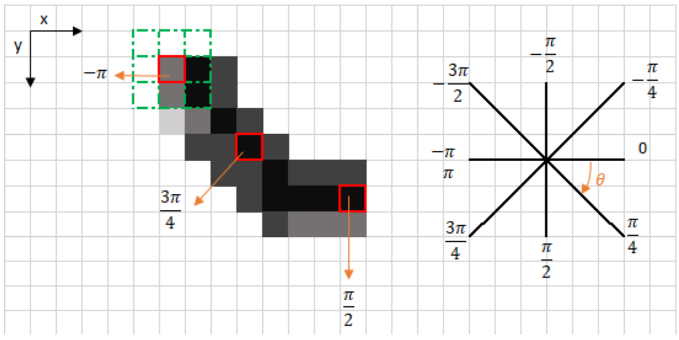


Figure 1.25: Example the edge directions[7].

An intensity pixel of the Gradient Intensity matrix being processed is shown by the red box in the top left corner of the (Figure 1.25 : example). The orange arrow with an angle of -pi Radians (+/-180 degrees) represents the matching edge direction. [7]

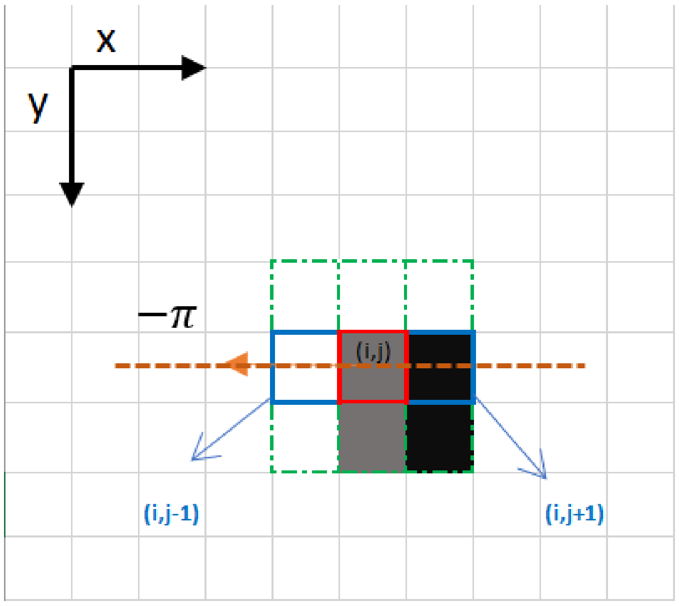


Figure 1.26: Focus on the upper left corner red box pixel[7].

The orange dotted line indicates the edge direction (horizontal from left to right). The algorithm's goal is to see if pixels in the same direction are brighter or darker than the ones being processed. The pixel (i, j) in (Figure 1.26 : Focus on the upper left corner red box pixel) is being processed, and the pixels in the same direction (i, j-1) and (i, j+1) are highlighted in blue. Only the more intense of those two pixels is maintained if one of them is more intense than the one being processed. Because it is white, pixel (i, j-1) appears to be more intense. As a result, the current pixel's intensity value (i, j) is set to zero. If there are no pixels with more intense values in the edge direction, the current pixel's value is preserved. [7] Edge Direction in Radians and Pixel Intensity (between 0 and 255) are the two most important characteristics for each pixel. Non-Max-Suppression steps are based on these inputs:
- Using the angle value from the angle matrix, determine the edge direction.
- Check if the pixel in the same direction has a higher intensity than the one being processed right now.

- Return the picture that was suppressed using the non-max technique.
[7] The following (Figure 1.27 : Non Maximum Suppression) is the output of the image:

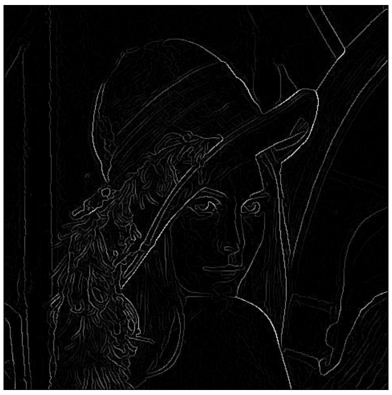


Figure 1.27: Non Maximum Suppression [6].

1.9.4 Double Threshold:

We can see that the result of Non-Maximum Suppression isn't ideal; certain edges may not be true edges, and the image has some noise. This is handled by Double Thresholding. [7] The Double Threshold step seeks to distinguish between three types of pixels: strong, weak, and irrelevant:
- Strong pixels are those with such a high intensity that we can be certain they influence to the final edge.
- Weak pixels are those with an intensity value that isn't high enough to be called strong, but not low enough to be regarded irrelevant for edge detection.
- Other pixels are ignored since they are irrelevant to the edge.
For the following, the double thresholds apply:
- High threshold: is used to find the pixels that are very bright (intensity $>$ high threshold).
- Low threshold: is used to find pixels that aren't relevant (intensity $<$ low threshold).
- All pixels with an intensity between the two thresholds are labeled as weak, and next step will assist us distinguish between those that are potentially strong and those that are irrelevant. [7]
The result of this step is (Figure 1.28: Double Thresholding):

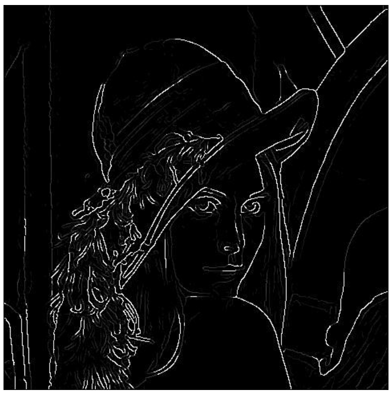


Figure 1.28: Double Thresholding [6].

1.9.5 Edge Tracking by Hysteresis:

The hysteresis transforms weak pixels into strong ones based on the threshold findings, but only if at least one of the pixels around the one being processed is a strong one, as stated below (Figure 1.29 : Edge Tracking by Hysteresis):

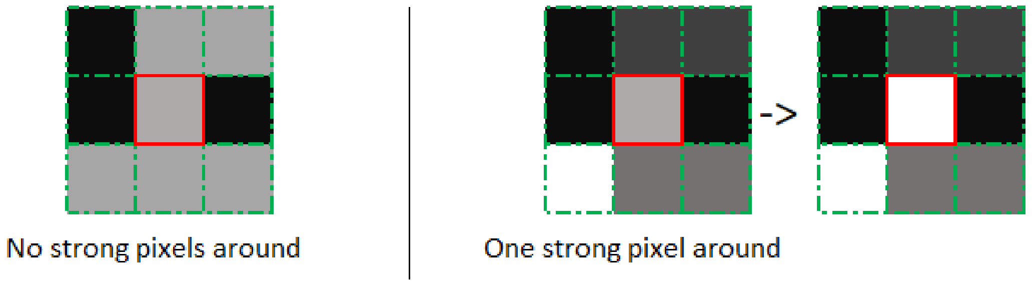


Figure 1.29: Edge Tracking by Hysteresis [7].

The result of this step is (Figure 1.30 : Results of hysteresis process):

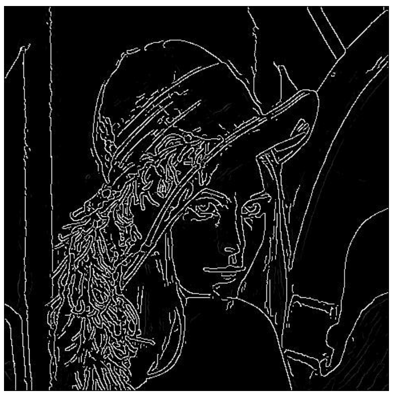


Figure 1.30: Results of Hysteresis Process [6].

1.10 Conclusion

In this chapter, the basic concepts are covered as Spam and Site we focus on it Twitter. And the general definition of approach we will use it as Bio-inspired algorithm and Artificial Neural Networks. Now we can get into more detail about our work.

Chapter 2

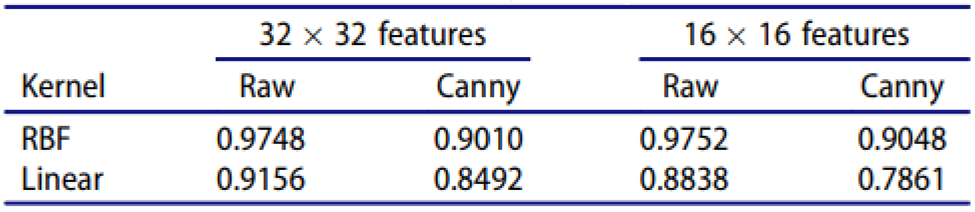
State of arts

2.1 Introduction

In our time, the amount of information and tweets are increasing. Twitter has been growing steadily, and studies have suggested that Twitter users share about 8.3 million tweets every hour. Unfortunately, Twitter is a popular place for "Spammers", which they publish unwanted message that may contain malicious software, advertisements, or links that contain malicious sites, and Spammers spread across different forms. Spammers insert their messages into images to avoid detection the spam. Previous image spam detection research has found that certain kinds of image spam can be detected with a level of accuracy focused on techniques of machine learning. In this chapter we focus on the various Machine Learning and Neural Network approaches, we go through Support Vector Machines (SVM), Multilayer Perceptrons (MLP), and Convolutional Neural Networks (CNN). There is also a discussion of the Particle Swarm Optimizer (PSO). It also discusses how to apply this approach with a Convolutional Neural Network to discover the appropriate hyperparameters and create a fully trained CNN architecture for a certain Dataset.

2.2 The Support Vector Machine (SVM) experiment :

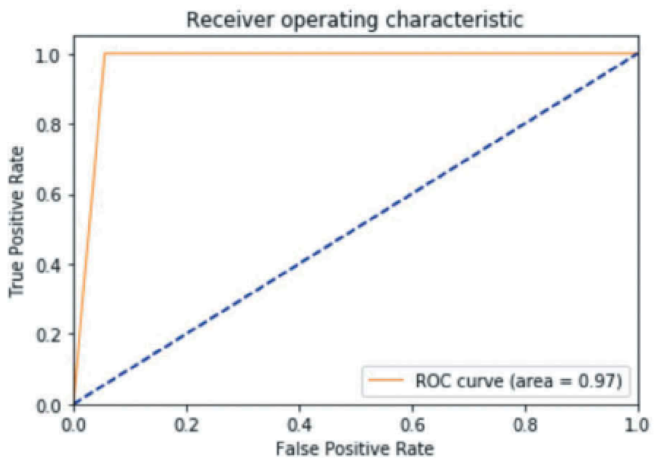
To train the models for the SVM experiments, he first create feature vectors. Images in the datasets are of various sizes. As a result, he resizes all of the photos to 32*32. He then converts a raw image into a Canny image using the Canny edge detection method. He generates byte data for each pixel in the Canny image to form the feature matrix. Each pixel is made up of three bytes that represent the red, green, and blue (RGB) color information in the 0 to 255 range. Each integer is normalized to be in the range of 0 to 1 for computational convenience. He also created a feature vector based on the raw byte values (also normalized).[5] Each feature vector has a length of 1024. He created unique SVM models for the dataset for the studies. He uses a random shuffle in each dataset and 70 percent of the image samples for training and the remaining 30 percent for testing. He uses both linear and RBF kernels, as well as various features, in his SVM research. In the figure2.1(SVM feature size and type comparison (ISH dataset)) : compares the accuracy of the SVM when trained and assessed on the ISH dataset with raw images shrunk to 32 32 as opposed to images resized to 16 16. He achieves a best accuracy of 0.9752 while employing the RBF kernel, which is significantly better than the best case for the linear kernel, which is 0.9156.The raw picture feature performs better in both circumstances than the Canny image feature.



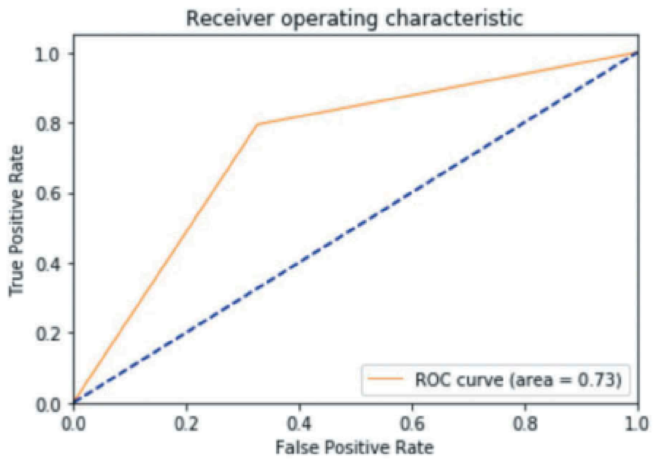
	32 × 32 features		16 × 16 features	
Kernel	Raw	Canny	Raw	Canny
RBF	0.9748	0.9010	0.9752	0.9048
Linear	0.9156	0.8492	0.8838	0.7861

Figure 2.1: SVM feature size and type comparison (ISH dataset) [5].

The difference between the two feature sizes in the RBF kernel is minor. figure2.2(ROC curves for ISH dataset.) : displays the ROC curves for both the RBF and linear kernels for the SVM binary classification results based on the ISH dataset. The RBF kernel has an area under the ROC curve of 0.97, while the linear kernel has an area under the ROC curve of 0.73. These findings show that an SVM using an RBF kernel can accurately distinguish ham and spam images with a low false positive rate.[5]



(a) RBF Kernel

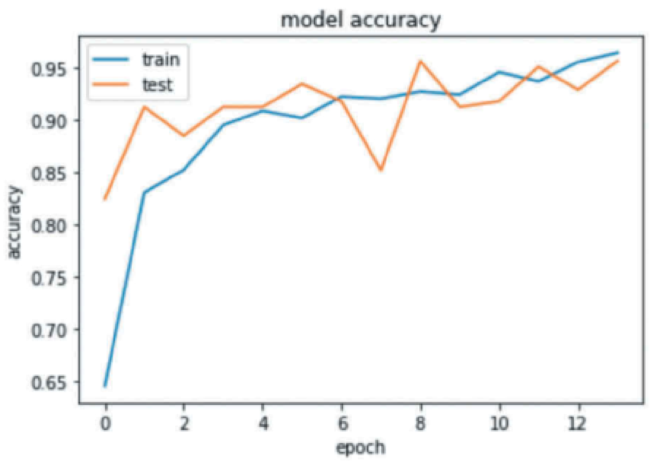


(b) Linear Kernel

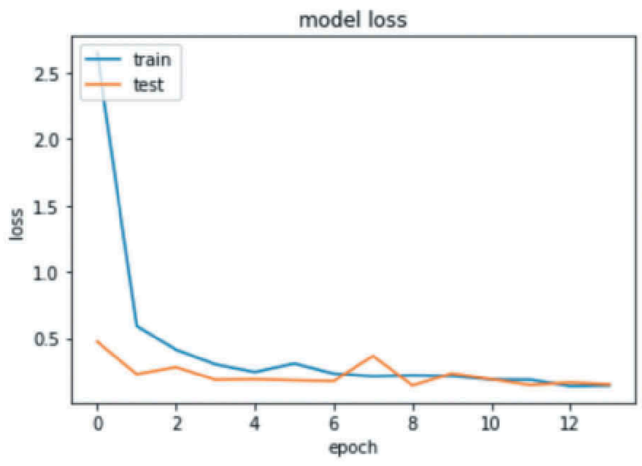
Figure 2.2: ROC curves for ISH dataset [5].

2.3 The Multi-Layer Perceptrons (MLP) experiment :

He experimented with many topologies for multilayer perceptrons (MLP). MLPs with one input layer, two hidden layers, and one output layer are studied in this paper. Each hidden layer has 128 nodes and the activation functions are rectifier linear units (ReLU). He chose the binary cross-entropy function to calculate the loss. At the output stage, a sigmoid score function is applied. The greatest results were routinely obtained with this architecture. The models are trained for 100 epochs and 70 % of the picture data are used to train the MLPs. A batch size of 64 is employed for each epoch, and the validation split is 15 percent of the picture data samples. figure2.3(MLP accuracy and loss) : demonstrate MLP accuracy and loss over 50 epochs, with figure 2.3(b)(MLP loss) showing the related loss graph.



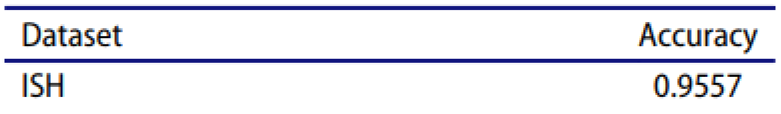
(a) Accuracy for ISH dataset



(b) Loss for ISH dataset

Figure 2.3: MLP accuracy and loss [5].

The model is converging without overfitting, according to these results. The ideal testing accuracies for the MLP experiments reported in figure2.4(MLP result).[5]



Dataset	Accuracy
ISH	0.9557

Figure 2.4: MLP result [5].

2.4 The Convolutional Neural Networks (CNN) experiment:

For image processing, convolutional neural networks are generally favorable - in terms of efficiency and accuracy. In the data set ISH. The results is based on a combination of characteristics (raw and Canny image). A variety of CNN hyperparameters have been experimented with, but for the purpose of this study the following setup is used. The first convolution layer has 32 nodes and 3 kernels. There are three convolutional layers, with 32 and 64 nodes in the second and third layers, respectively. Use the ReLU activation function on all convolutional layers, and the sigmoid function on the fully connected final layer. Use a pool size of 2 2 to downsample the data using a maximum pooling layer. A 0.5 dropout rate was also used to avoid overfitting.For each epoch, the batch size is set to 64, training for 100 epochs using 70 % of the data used for training and 30 % retained for testing. figure2.5(CNN architecture) : shows the architecture of a CNN network.

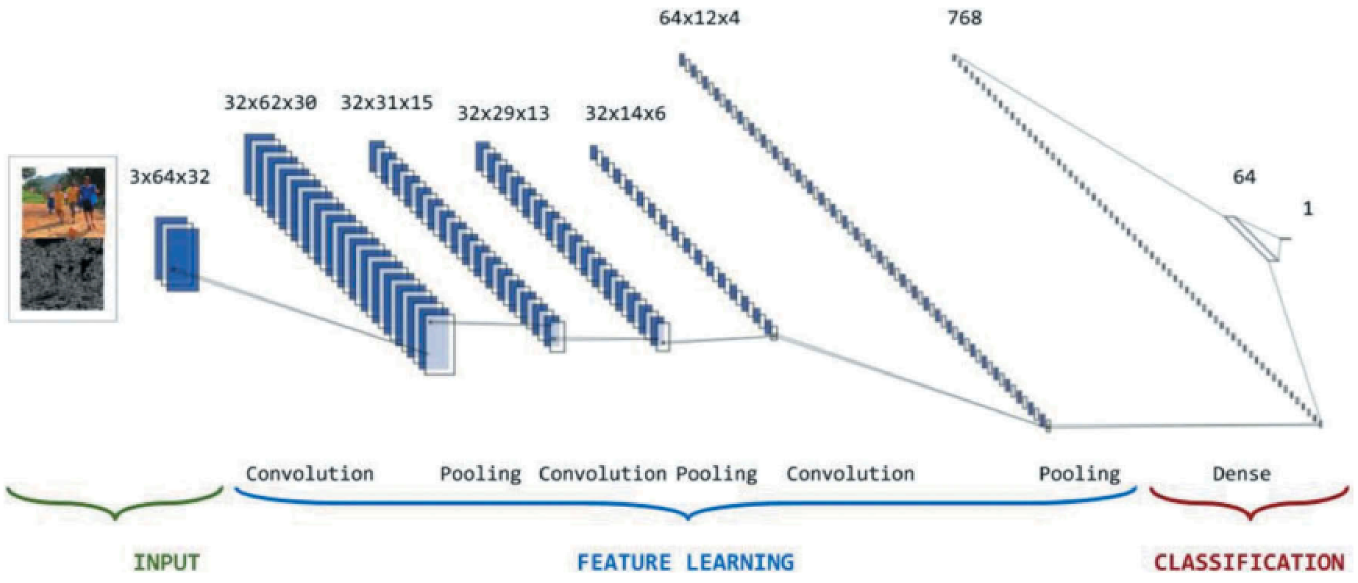
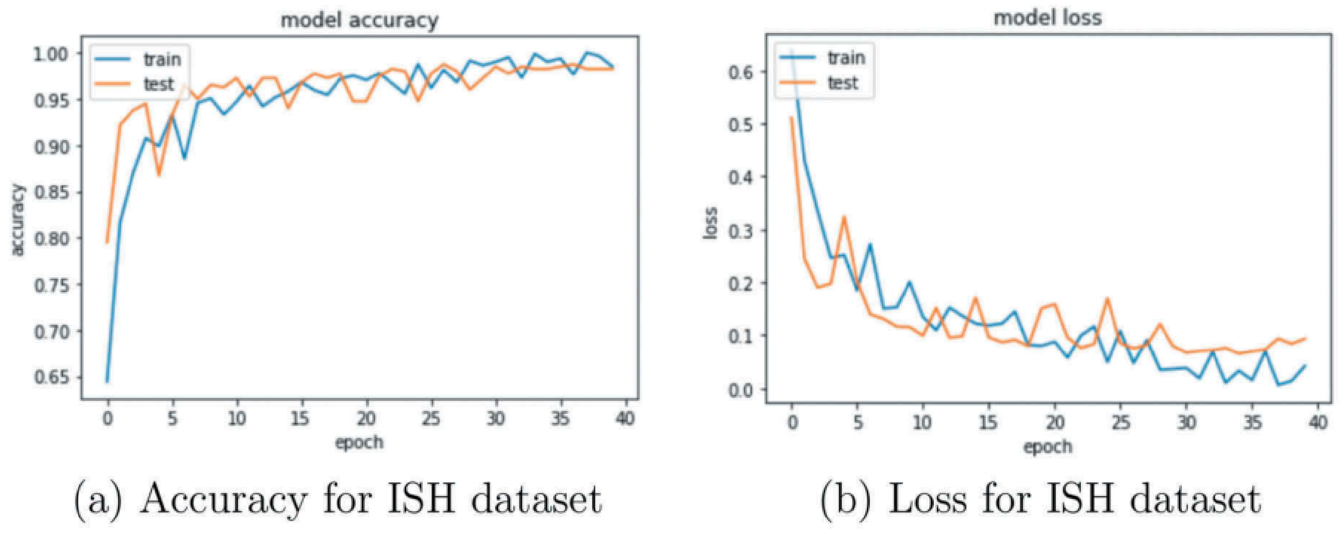


Figure 2.5: CNN architecture [5].

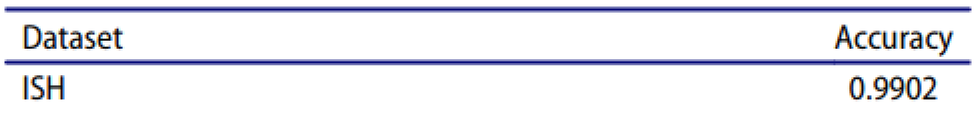
Figure2.6(CNN accuracy and loss) : shows the accuracy and loss graphs of the ISH data set, which clearly show that no overfitting occurred. The heights in the test graphs for the challenge data set indicate the difficulty the models have with the data; Even a small advance in the training set can lead to instability in the validation set.



(a) Accuracy for ISH dataset (b) Loss for ISH dataset

Figure 2.6: CNN accuracy and loss [5].

It may be possible to mitigate these mutations by using more regulation (eg, dropouts), but this would have little effect on outcomes and would increase the cost of training significantly. Figure2.7(CNN result) : shows the best CNN test accuracy for the data set under consideration. In the ISH dataset, we can observe that CNN outperforms both SVM and MLP.[5]



Dataset	Accuracy
ISH	0.9902

Figure 2.7: CNN result [5].

2.5 Particle Swarm Optimization (PSO) :

Optimization of Particle Swarms (PSO) It's a Meta-Heuristic optimization strategy based on population. It has attracted the attentions of researchers all around the world due to its applicability and performance in a wide range of complicated real-world optimization issues. In 1995, Kennedy and Eberhart created the Particle Swarm Optimization technique. It is inspired by the way a flock of birds in search of food sources adjusts their position based on their individual previous positions as well as the position of their swarm. It starts by creating a number of discrete search 'Particles' each representing a potential solution. An evolutionary process causes this population of particles to shift their places.[23]

2.5.1 How PSO works:

Simple mathematical calculations over the particle's position and velocity are used to move these particles around in the search space. The particles' movements (Figure 2.8 : Movement of Particle) are directed by their previous best position in the search space (Personal Best) and also the swarm's best-known position (Global Best).[24]
It requires finding an optimal location within a specific neighborhood, which is determined by three factors: inertia: the particle tends to follow its current direction of movement cognitive: the particle tends to go to the best site by which it has already passed social: the particle tends to rely on the experience of its congeners and, thus, to move towards the best site already reached by its neighbors. [24]

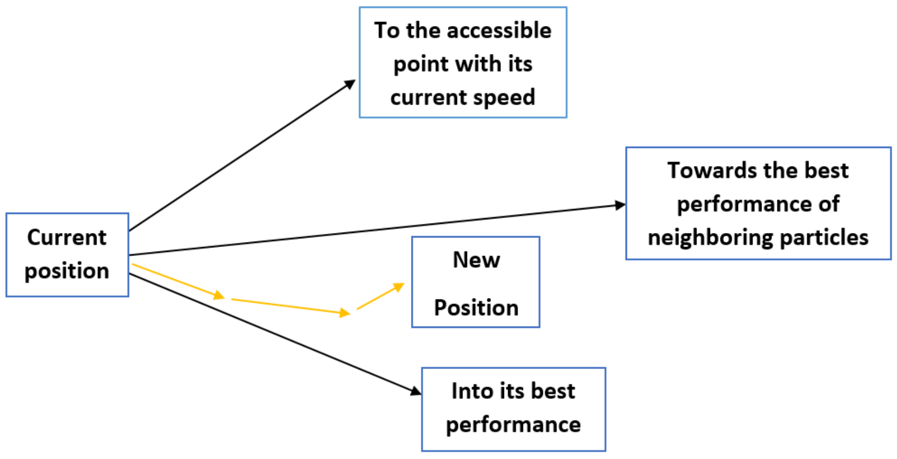

To the accessible point with its current speed
Towards the best performance of neighboring particles
Current position
New Position
Into its best performance

Figure 2.8: Movement of a particle.

Components of PSO:

The algorithm's procedure then requires moving these particles in order for them to discover the best solution; they must have:
-their locations, knowing their coordinates under the condition that they are part of the search space
-the best position they've ever had.
-Their neighborhood's optimal position, as determined by their objective function.
-Their speed, which enables them to move and change positions throughout the iterations.
-a neighborhood, which is a group of particles that communicates with the particle directly (especially the one with the best position).[25]

2.5.2 The Concept Of The Neighborhood :

The particle's neighborhood is made up of a group of other particles with whom it interacts. The interweaving of all the particles' relationships is referred to as sociometry or swarm topology. There are two types of neighborhoods:
- Geographic Neighborhood: it is a dynamic neighborhood in which the closest particles are the neighbors. The new neighbors or groups must be revised at each iteration by reference to a specified distance in the research space. [25]

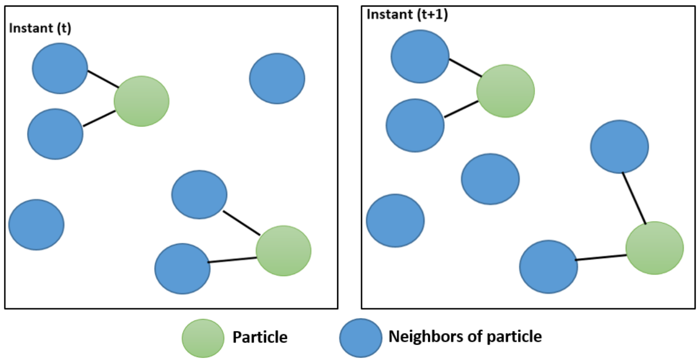


Figure 2.9: Geographical Neighborhood at (t) and (t + 1).

The concept of dynamic neighborhood is illustrated in (Figure 2.9: Geographical neighborhood at (t) and (t + 1) the neighborhood for the same swarm at (t) and (t + 1) is not the same. Note: in this example, we'll suppose that a particle's neighborhood is composed of the two closest particles.
- Social Neighborhoods: These neighborhoods are considered static because they do not change. This is the most commonly used neighborhood because of:

- In terms of calculation, it provides a better time/cost ratio.
- In a convergence scenario, a social neighborhood is closely connected to a geographic neighborhood.
- Its programming simplicity. [25]

2.5.3 Number Of Particles:

The dimension of the search space and the relation between the machine's processing capabilities and the maximum time complexity determine the number of particles that used fix the issues. There is no law that defines this parameter; nonetheless, and do a large number of tests will provide you with the essential expertise to understand it. The decision is made randomly in most cases. [26]

2.5.4 Neighborhood Topology:

The structure of the social network is defined by the topology of the neighborhood, which determines with whom each particle will be able to communicate. There are other combinations, but the following are the most popular (Figure 2.10: Neighborhood Topologies):

- Star Topology: each particle is connected to all others, and the best global is the best neighborhood.
- Ring Topology: each particle is linked to n other particles (usually n = 3), and each particle tends to move toward the best in its near area.
- Radius Topology: just one central particle connects with the particles.[26]

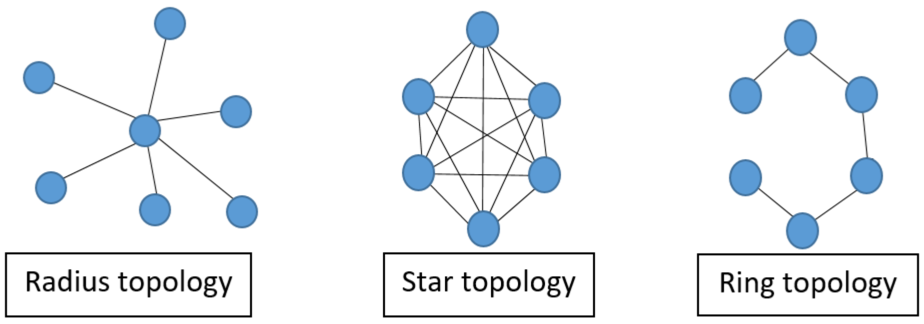


Figure 2.10: Neighborhood Topologies Most Used.

2.5.5 PSO Algorithm:

A particle i in the swarm is represented by its position vector and velocity vector in the D-dimension research space, as follows:

$$P_i^t = [x_{0,i}^t, \quad x_{1,i}^t, \quad x_{2,i}^t, \quad x_{3,i}^t, \quad \ldots, \quad x_{n,i}^t]$$

In a research space, a group of particles (possible solutions) of the global minimum. None of the particles knows where the global minimum is, but they all have fitness values that need to be optimized using the fitness function. [8]

$$V_i^t = [v_{0,i}^t, \quad v_{1,i}^t, \quad v_{2,i}^t, \quad v_{3,i}^t, \quad \ldots, \quad v_{n,i}^t]$$

Each of these particles has a velocity that allows them to update their position over time in order to discover the global minimum.[8] In the search space, the particles have already been scattered at random. After that, their velocity must be set. The velocity vector, which is defined by its speed in each direction, will be randomized once more. [8] determine the particle's speed, which then allows us to calculate the particle's position by using equation (Figure 2.11: Particle Update).[8]

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

$$V_i^{t+1} = \underbrace{wV_i^t}_{\textbf{Inertia}} + \underbrace{c_1 r_1 (P_{best(i)}^t - P_i^t)}_{\textbf{Cognitive (Personal)}} + \underbrace{c_2 r_2 (P_{bestglobal}^t - P_i^t)}_{\textbf{Social (Global)}}$$

Figure 2.11: Particle Update.[8]

There are two aspects to our worth. The best personal solution and the best global solution, the first discovered by a single particle and the second by a swarm of particles. Each particle stores the best personal and global solutions in its memory. [8]
$(wV)_i^t$correlates to the displacement's inertia component, where the w parameter determines the influence of displacement direction on future displacement. $c_1r_1(p_(best(i))^t - p_i^t)$ relates to the cognitive component of displacement, with parameter c1 controlling the particle's cognitive behavior. $c_2r_2(p_bestglobal^t - p_i^t)$ corresponds to the social component of the displacement, with c2 controlling the particle's social fitness. The velocity of every particle is updated. The two best values found so far control this velocity, which is influenced by inertia. [25] Random terms are used to weight acceleration at each iteration. The weights r1 and r2 are used to change the cognitive and social accelerations stochastically. [8] Every particle and iteration have two distinct weights, r1 and r2. The hyperparameter w (Inertia Weight) balance between exploring and exploiting the greatest solutions discovered thus far, which allows to set the swarm's ability to shift direction. A low coefficient w makes it easier to use the best solutions found so far, which means It implies a higher level of convergence as illustrated in (Figure 2.12: W Comparison). [8] While a higher coefficient w makes it easier to explore these solutions.
Note:(w >1) should be avoided because it can cause our particles to diverge. [8]

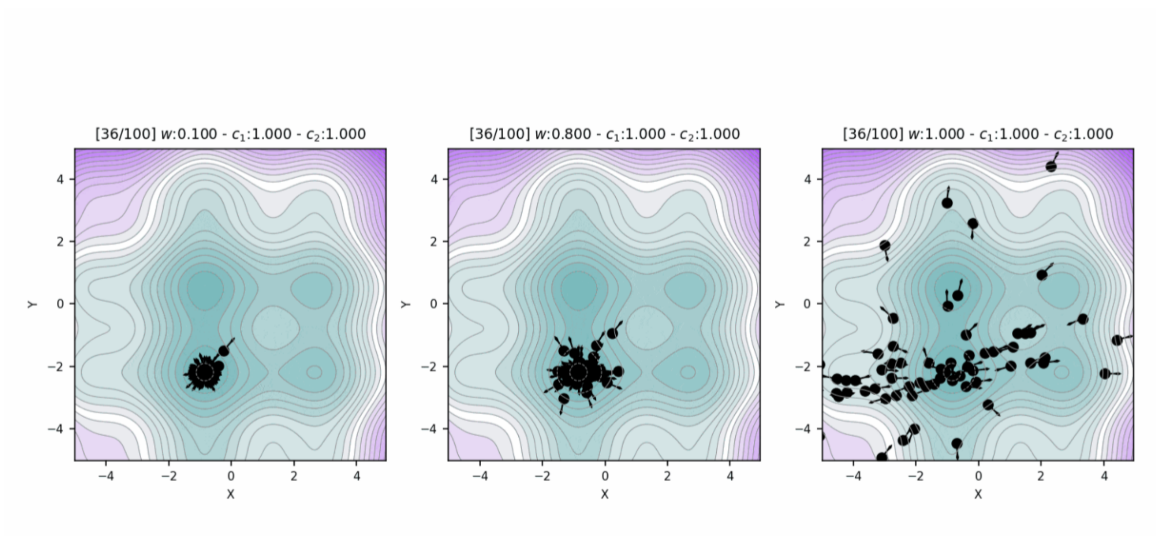


Figure 2.12: W Comparison.[8]

The C1, C2 (Acceleration Coefficients) hyperparameter defines the group's ability to be influenced by the best individual solutions for the first Coefficients and the best global solution for the second Coefficients obtained over iterations. [8]

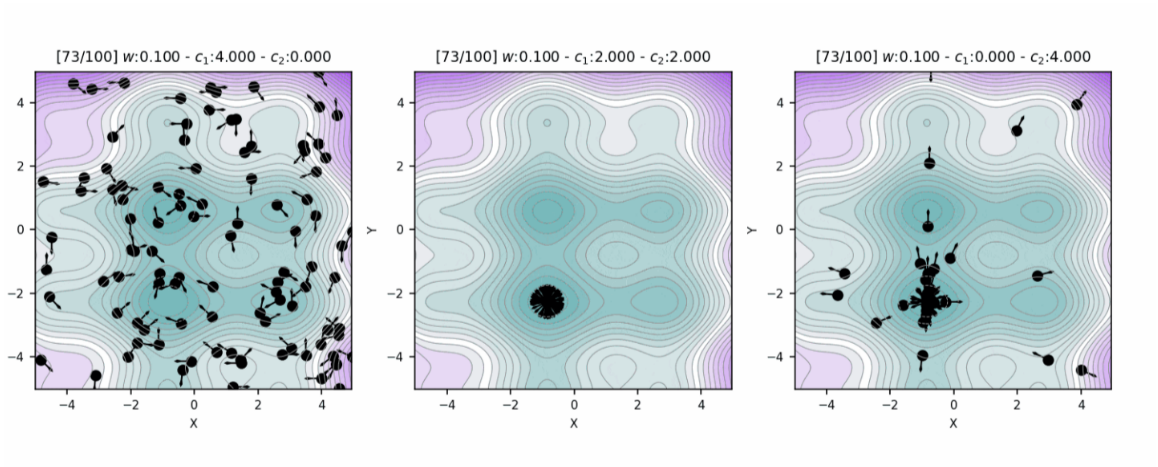


Figure 2.13: C1 and C2 Comparison.[8]

When C1 is high, we see that the swarm particles become more individualized, no convergence occurs because each particle is exclusively concerned with its own best solutions. When C2 is high, then, the swarm's particles will be more impacted by others as illustrated in (Figure 2.13 : C1 and C2 comparison). [8] As a result, the coefficients C1 and C2 are complimentary. Exploration and exploitation are both boosted when the two are combined. [8]
Algorithm:

Algorithm 1: Particle swarm optimisation algorithm

```
1. Initialize N particles at random with their positions and velocities.
while For each particle i do
    • Evaluate particle positions: fitness f (Pi) at current position Pi
    if f (Pi) is better than f (Pbest(i)) then
        update Pbest(i) and f (Pbest(i))
    end
    if f (Pi) is better than f (Pbestglobal(i)) then
        update Pbestglobal (i) and f (Pbestglobal (i))
    end
    For each particle i Update Velocity Vi and position Pi using: Figure: Particle update
end
```

Notation:
Pi: a vector denoting its position.
vi: the vector denoting its velocity.
f (Pi): denotes the fitness score of Pi.
Pbest(i): the best position that it has found so far.
f (Pbest(i)): denotes the fitness of Pbest(i).
Pbestglobal (i): the best position that has been found so far in its Neighborhood.
f (Pbestglobal(i)): denotes the fitness of Pbestglobal (i).

2.5.6 Auto Hyperparameters:

We can take it a step further by updating coefficients as we go through the iterations. Beginning with a strong C1, strong W, and weak C2 to enhance search space exploration, we want to converge to a weak C1, weak w, and strong C2 to exploit the best results after exploration by converging to the global minimum.[8]

$$w^t = 0.4\frac{(t-N)}{N^2} + 0.4$$

$$c_1^t = -3\frac{t}{N} + 3.5$$

$$c_2^t = +3\frac{t}{N} + 0.5$$

Figure 2.14: Auto Hyperparameters Over Iterations.[8]

The optimal static parameters are W=0.72984 and C1 + C2 > 4 Specifically, C1 = C2 = 2.05 according to M.Clerc and J. Kennedy's paper [27] to create a standard for Particle Swarm Optimization.[8]

2.5.7 Convolutional Neural Network Architecture Optimized by PSO :

This section discusses optimization techniques that use the PSO algorithm to improve the parameters of CNN architectures. After evaluating the performance of a CNN through an experimental study

in which the parameters were updated manually, the parameters to be optimized were picked.[9] For the same task, various CNN parameter choices provide a difference of outcomes; the goal is to find the best architectures. In this study, the parameters stated below were chosen to be optimized.
• The number of convolutional layers;
• The filter size or filter dimension used in each convolutional;
• The number of filters;
• The batch size number (the number of images that are entered; into CNN in each training block);
The general scheme is introduced in (Figure 2.15 : General CNN Optimization Process Using The PSO Algorithm).

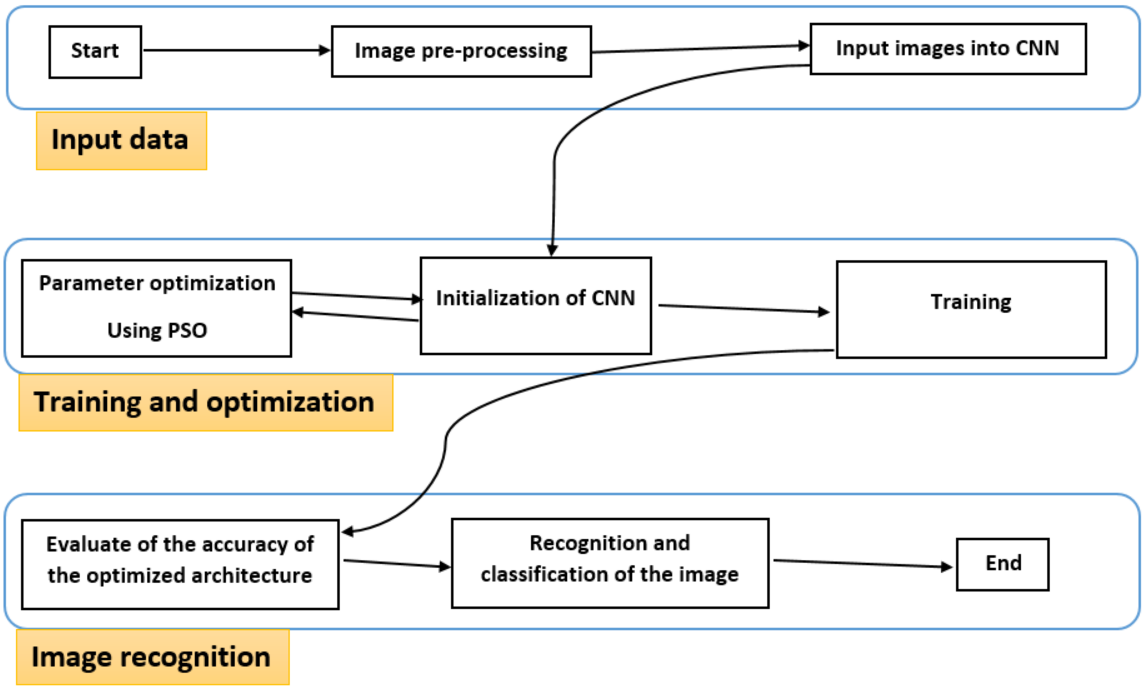

Start
Image pre-processing
Input images into CNN
Input data
Parameter optimization Using PSO
Initialization of CNN
Training
Training and optimization
Evaluate of the accuracy of the optimized architecture
Recognition and classification of the image
End
Image recognition

Figure 2.15: General CNN Optimization Process Using The PSO Algorithm. [9]

The PSO technique is used to integrate the parameter optimization into the CNN. The PSO is started per the execution parameter, and the particles are generated as a result. Each particle provides a complete CNN training since it offers a possible solution and each position has the parameter to be optimized.[9] The steps for using the PSO algorithm to optimize the CNN are depicted in (Figure 2.16 : Flowchart Of CNN Optimization Process Using PSO) and described in following:
• Input database: picking the database to be processed and classified.
• Generate the particle population for the PSO algorithm: The number of iterations and particles, inertial weight, cognitive constant (W1), and social constant (W2) are all PSO parameters that were employed in the experiment. The design of the particles is involved in this step.
• Construct the CNN architecture: The CNN is set and prepared to train the input database with the parameter given by the PSO.
• CNN training and validation.
• Evaluate the objective function: To obtain the best value, the PSO algorithm examines the objective function.

• Update PSO parameters: Every particle changes its velocity and position in the search space based on its best position (Pbest) and best position in the swarm (Gbest) during every iteration.
• The process continues until the stop condition are reached by evaluating all of the particles (number of iterations).
• Lastly, the best solution is chosen: (Gbest) represents a particle in this process.[9]

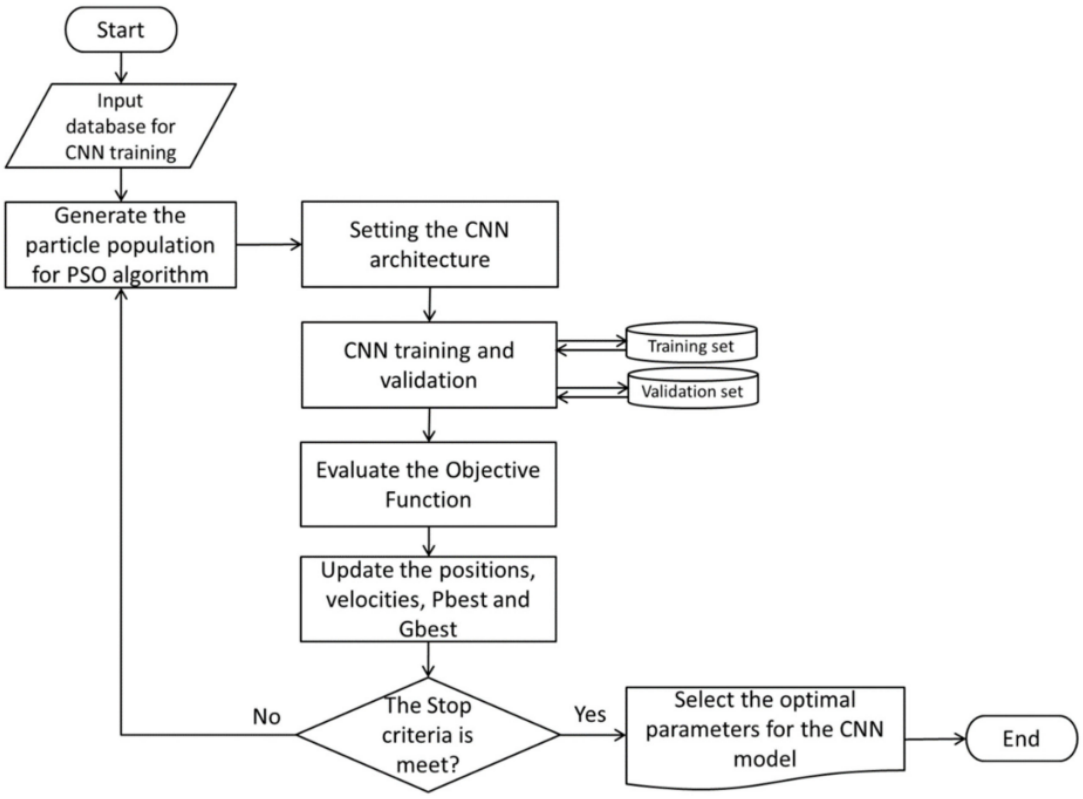

Start
Input database for CNN training
Generate the particle population for PSO algorithm
Setting the CNN architecture
CNN training and validation
Training set
Validation set
Evaluate the Objective Function
Update the positions, velocities, Pbest and Gbest
The Stop criteria is meet?
No
Yes
Select the optimal parameters for the CNN model
End

Figure 2.16: Flowchart Of CNN Optimization Process Using PSO .[9]

2.6 Conclusion

In this chapter we present different Learning Techniques experiments for Identifying Spam from Ham Images and its result and we did discussion this results. Support vector machines (SVM) and two Neural Network-based approaches, Multilayer Perceptron (MLP) and Convolutional Neural Networks(CNN), were explicitly addressed. Also the Particle Swarm Optimization (PSO) algorithm, which was inspired by the world of animals (bird species), has had exceptional success since its beginnings Due to its simplicity.

Chapter 3

Implementation

3.1 Introduction :

In this chapter, we present an experimental part. When trying to design CNN architectures, we face several challenges, such as high computational costs for information processing and determining the best CNN parameters (architecture) for every issue. To address these issues, we will recommend the PSO algorithm, which is used to automatically design CNN architectures in order to get best result

3.2 Runtime Environment :

In this section, we'll go over the software and hardware that make up the development environment of our implementation's.
• Python: is the most ideal programming language since it is utilized in artificial intelligence. It has made significant advancements in programming, and is recognized an accurate, strong, and simple to learn language. Python is also a wonderful approach to apply deep learning to classify images with using some library.
• Keras: is a high-level neural networks simple and powerful free open source Python library that allows easy and fast prototyping it is for defining and training practically all type of deep learning model. Tensorflow, Theano, and CNTK are all able to operate on top of it. It was created to allow for quick experimenting.
• Google Collab: is a Google Research product. Colab is a web-based Python editor that allows anyone to write and run arbitrary Python code. It's notably useful for machine learning, data analysis, and education. Colab is a hosted Jupyter notebook service that requires no installation and provides free access to computer resources such as GPUs. with Intel(R) Xeon(R)Intel(R) Xeon(R) CPU @ 2.30GHz, about 13 GB of RAM.

3.3 Image Spam Dataset :

There are only a few of publicly available image spam databases. As part of this study, we looked at some:
• ISH Dataset (Image Spam Hunter): Researchers at Northwestern University collected this data[28]. There are 928 spam photos as the spam sample collection, taken from real spam emails and 810 ham images 20 scanned documents were randomly collected from "Flickr.com" in this dataset and is

available at [29].
• Dredze Dataset: Dredze et. al [30] created an image spam corpus which is publicly available [31]. The data set consists of a large corpus with different formats (JPEG, GIF, PNG). We used a simple criterion: if an image appears in a spam email, it is spam. It's ham if it appears in a ham message. We use the standard definitions of spam and ham email. The data is clearly labeled according to this standard. created two spam datasets, one depending on individual spam and the other depending on spam that was publicly available. Over the course of a month, we collected spam emails from 10 email accounts across 10 domains. A two-year period of time was used to collect mails and extract all accompanying images for ham. Our ham images are more realistic because they come from actual user emails. Due to the absence of public email image data, it was unable to create a public ham corpus.
• Challenge Dataset 1: The authors of [32] developed this challenge dataset and available in [33]. They use image processing techniques to spam images to make them look as ham-like. The spam images were taken from the Dredze spam repository public corpus [31]. This dataset contains 1000 images. From the spammer's perspective, this dataset has been "enhanced.". These spam images were mixed with the ham images from the ISH [29] dataset using a weighted overlay technique in a suitability study, a weighted overlay is the intersection of standardized and differently weighted layers. The weights represent the relative relevance of the various suitability criteria. [10]

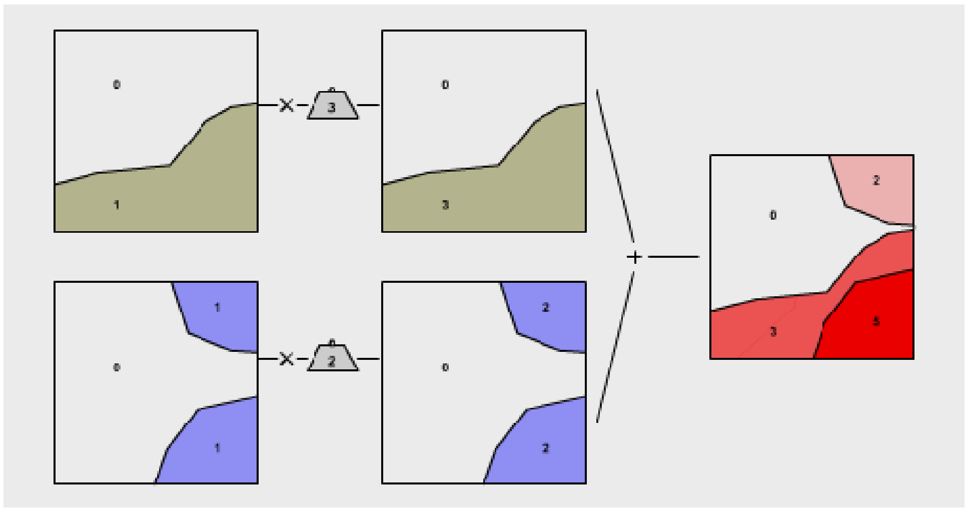


Figure 3.1: Weighted overlay. [10]

• Challenge Dataset 2: The dataset for this challenge was created as part of a study in [34]. the objective of this dataset is to provide a challenge to the detection of increasingly advanced kinds of image spam that are sure to be observed in the coming days employing an alternative method of overlaying like in [32]. ISH [29] and the Dredze [31] datasets were used for our spam corpus. In order to create our spam images, we first extract the content of an existing spam image, then overlay it on an existing ham, (Figure 3.2 : Challenge dataset example) illustrates an example of a spam image created using The modified spam image like a ham.

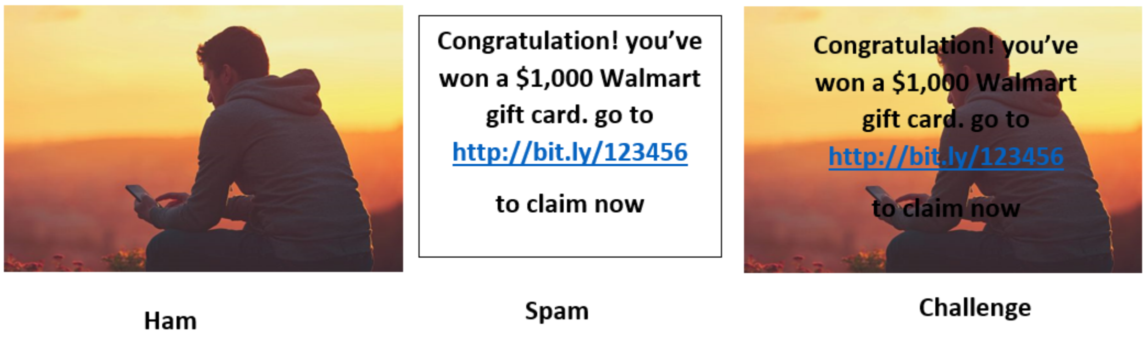
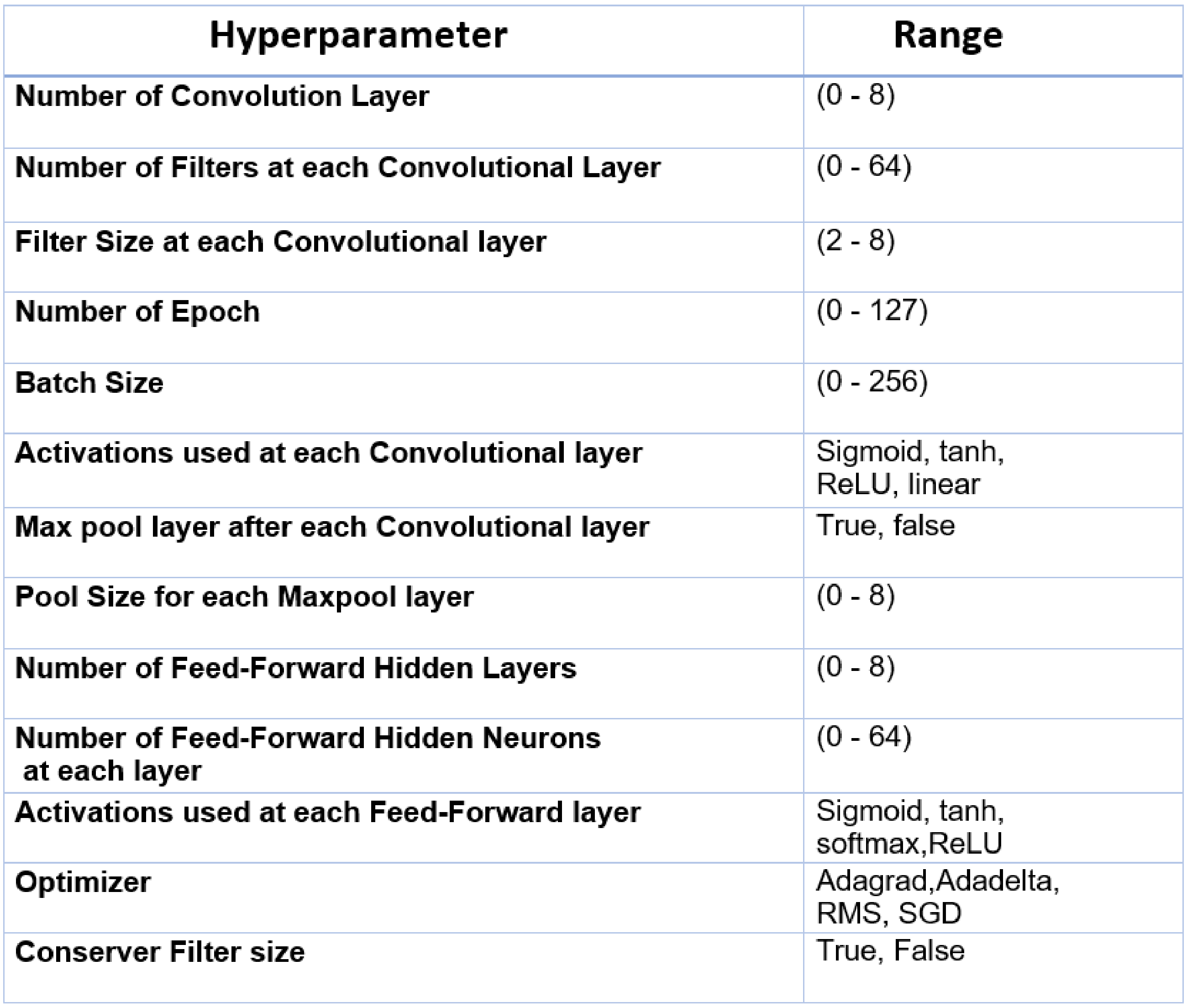

Congratulation! you've won a $1,000 Walmart gift card. go to http://bit.ly/123456 to claim now

Figure 3.2: Challenge Dataset Example.

3.4 Tuning CNN Hyperparameters Using PSO :

Y ou must first choose the hyperparameters related with each layer of a deep neural network before you can begin training it. In most cases, human logic, experience, or trial and error determines the hyperparameters. Hyperparameters connected with deep neural networks and their ranges are listed in Table below (table 3.1 : The various hyperparameters in CNN with their ranges).[24]



Hyperparameter	Range
Number of Convolution Layer	(0 - 8)
Number of Filters at each Convolutional Layer	(0 - 64)
Filter Size at each Convolutional layer	(2 - 8)
Number of Epoch	(0 - 127)
Batch Size	(0 - 256)
Activations used at each Convolutional layer	Sigmoid, tanh, ReLU, linear
Max pool layer after each Convolutional layer	True, false
Pool Size for each Maxpool layer	(0 - 8)
Number of Feed-Forward Hidden Layers	(0 - 8)
Number of Feed-Forward Hidden Neurons at each layer	(0 - 64)
Activations used at each Feed-Forward layer	Sigmoid, tanh, softmax,ReLU
Optimizer	Adagrad,Adadelta, RMS, SGD
Conserver Filter size	True, False

Table 3.1: The various hyperparameters in CNN with their ranges.

3.5 Preparing The Data :

O ur experimental data is presented in this section, which begins with a description of the data set that we have chosen for our experiment. Onto the preprocessing step of this dataset, which is a really intriguing part of any data study project. The dataset we use is a collection of Spam Images and Natural Images. This dataset is downloadable from the following link:[35] The dataset includes 808 of Natural Images and 929 of Spam Images we split the dataset into training and validation subset (Listing A.1 – Data Generation). After splitting dataset, we Found 1356 images (training data) and 346 images (validation data) belonging to 2 classes.

3.6 Class Particle :

I n each population, we initialize particle (Listing A.2 – Initialization Particle) with this parameter (min-layer, max-layer, max-pool-layers, input-width, input-height, input-channels, conv-prob, pool-prob, fc-prob, max-conv-kernel, max-out-ch, max-fc-neurons, output-dim). aim to Build particle architecture, we Initial velocity and then we Update Particle. (Listing –A.3 Update Particle) and compute Velocity (Listing –A.4 Compute Velocity) during the training. compile our CNN model (Listing – A.5 Compile model), we start by create an object of the sequential class to initialize our neural network model as a sequential network, we import $Conv2D$ this is to perform the convolution operation on the training images. Since we are working on images here, which a basically 2 Dimensional arrays and $MaxPooling2D$, $AveragePooling2D$, $MaxPooling2D$ which is used for pooling operation, Dense which is used to perform the full connection of the neural network, after that we fit the data to our model (Listing – A.6 Fit the model).

3.7 Class Utils :

I n this class, we define many functions as:
- add-conv: Use it to add convolution layer (Listing –A.7 add convolution) we pass to list layer,Maximum output channel, convolution kernel and return list of layers.
- add-fc: Use it to add fully connected layer we pass list layer and maximum fully connected neurons, return list of layers.
- add-pool: Use it to add fully connected layer we pass list of layer, maximum fully connected neurons, return list of layers.
- differenceConvPool: Compute the difference only between the convolution and pooling layers.
- differenceFC: Compute the difference between the fully connected layers.
- computeDifference: Compute the difference between the best layers in particle or global best and layers.
- velocityConvPool: Compute the velocity only between the convolution and pooling layers.
- velocityFC: Compute the velocity between the fully connected layers.
- update functions for ConvPool, FC (fully connected), Particle.

3.8 Generate The Particle Population :

I n this class(Listing –A.7 Population class) will create N particles with random CNN architectures,. Each particle will have a random number of layers, but the first and end layers of each particle will always be a convolution and a fully-connected layer, respectively, in order to generate workable CNN architectures. Furthermore, fully-connected layers (FC) can only be used towards the end of the architecture, not between convolutions or pooling layers.

3.9 Optimize CNN Architecture by PSO :

T his class as shown in (listing :A.9 PSOCNN) presents the optimization approaches where the PSO algorithm is applied to optimize the parameters of CNN architectures is the main purpose of this the method , After searching and performed The fitness evaluation, the optimal architecture can be used to classify all the data set.

3.10 The Main Method :

I n the main method as presented in the (listing :A.10 The main method) , we initialize all the parameters that we need for the psoCNN method and call this method, and after that we display the details of the results of our optimizer architecture.

3.11 Discuss The Results :

T he accuracy and loss graphs for the ISH dataset for simple CNN architecture are given in (Figure 3.3: CNN Lose and Accuracy) and for PSO-CNN architecture are given in (Figure 3.4: PSO-CNN Lose)

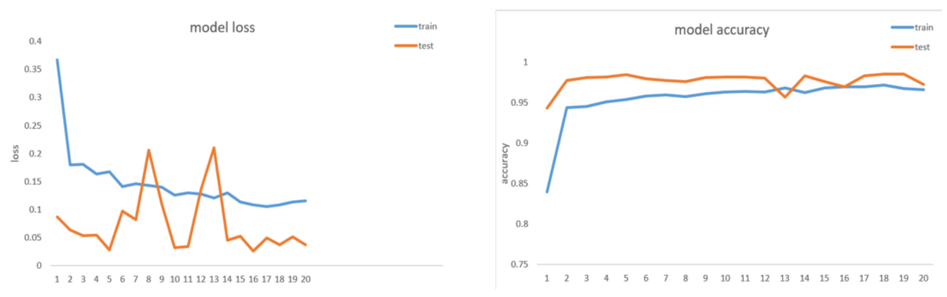


Figure 3.3: CNN loss and accuracy

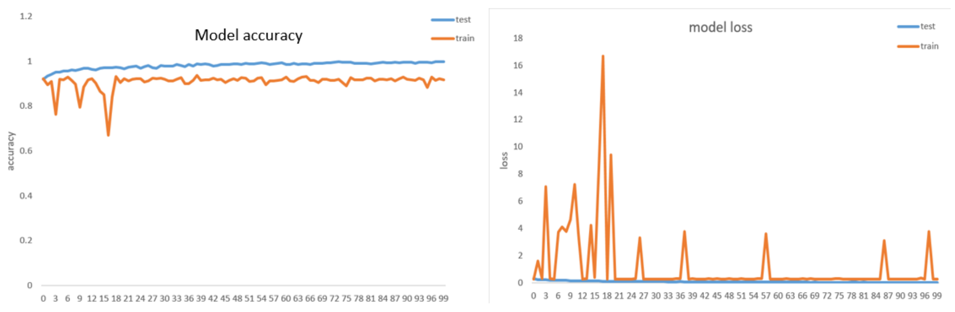
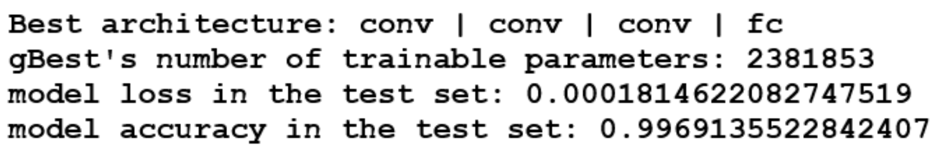


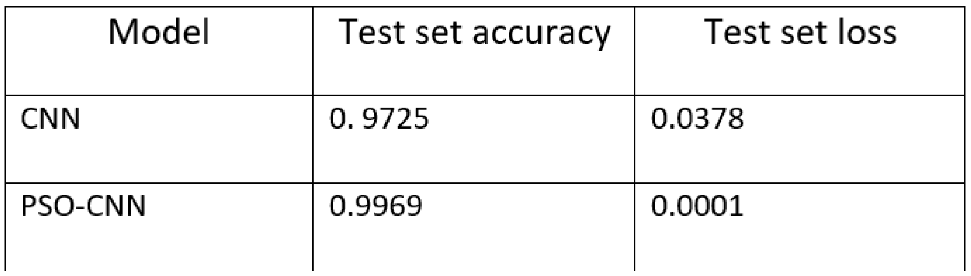
Figure 3.4: PSO-CNN loss and accuracy

According to (Figure:3.5 the result), the model performed well throughout the test the accuracy rate was 99.6%and the error rate was 0.018% and number of trainable parameters: 2381853 , indicating that it is stable (no sign of over-fitting) because it performed well during the test.
The optimal CNN testing accuracies for the datasets under consideration are given in (Figure:3.6 PSO-CNN and CNN Result). From these results, we see that our PSO-CNN performs better than simple CNN in ISH dataset.



```
Best architecture: conv | conv | conv | fc
gBest's number of trainable parameters: 2381853
model loss in the test set: 0.0001814622082747519
model accuracy in the test set: 0.9969135522842407
```

Figure 3.5: The Result



Model	Test set accuracy	Test set loss
CNN	0. 9725	0.0378
PSO-CNN	0.9969	0.0001

Figure 3.6: CNN and PSO-CNN Result

the architecture of model PSO-CNN in illustrated in (Figure:3.7 PSO-CNN Architecture).

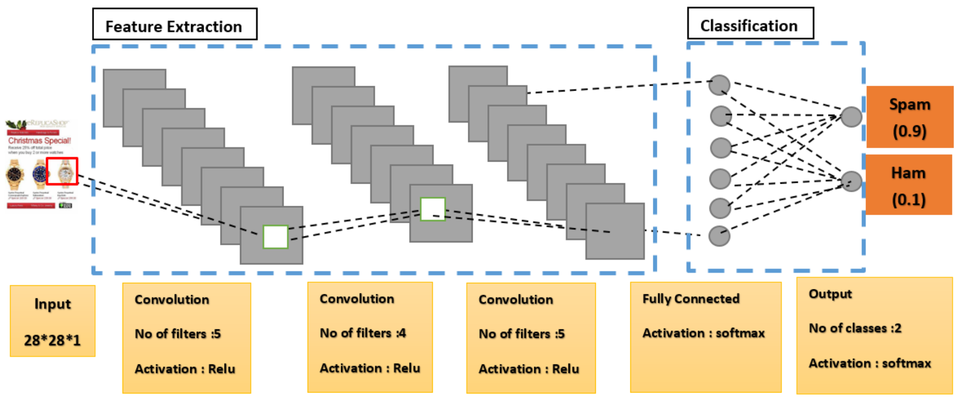


Figure 3.7: PSO-CNN Architecture

3.12 Conclusion :

In this chapter, we describe the procedures we took in the Experimental study to construct a spam model based on a data collection of image spam and natural images. Our model performed well, with an accuracy of 98 percent and a 3The results were satisfactory, but we believe that they may be improved further if we used a data set with a large number of images.

General Conclusion

In this thesis, we focus in image spam detection using a deep learning approach and bio-inspired algorithms. We started with Artificial Neural Networks, bio-inspired algorithms, Twitter, and meta-heuristics, with such a focus on Convolutional Neural Networks we employed it to build our model before moving on to PSO to generate fully trained CNN architectures for the (spam image /natural picture) Dataset. The proposed method has been shown to be capable of choosing important hyperparameters and building an optimal CNN architecture. We would have preferred working on a larger and more important data set, but the absence of data structure on Image Spam stopped us from doing so.

Appendix A

Python code to solve the optimization problem

```python
from keras.preprocessing.image import ImageDataGenerator
import keras

train_datagen = ImageDataGenerator(rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    validation_split=0.2) # set validation split

train_generator = train_datagen.flow_from_directory(
    r"/content/drive/MyDrive/datasetsIHM",# dataset path in drive
    target_size = (28, 28),
    color_mode="grayscale",
    shuffle=True,
    batch_size=32,
    class_mode='categorical',
    subset='training') # set as training data

validation_generator = train_datagen.flow_from_directory(
    r"/content/drive/MyDrive/datasetsIHM", # dataset path in drive
    target_size = (28, 28),
    color_mode="grayscale",
    batch_size=32,
    shuffle=True,
    class_mode='categorical',
    subset='validation') # set as validation data
```

Listing A.1: Data Generation

```python
class Particle:
    def __init__(self, min_layer, max_layer, max_pool_layers, input_width,
   input_height, input_channels, \
        conv_prob, pool_prob, fc_prob, max_conv_kernel, max_out_ch,
   max_fc_neurons, output_dim):
        self.input_width = input_width
        self.input_height = input_height
        self.input_channels = input_channels

        self.num_pool_layers = 0
        self.max_pool_layers = max_pool_layers

        self.feature_width = input_width
        self.feature_height = input_height

        self.depth = np.random.randint(min_layer, max_layer)
        self.conv_prob = conv_prob
        self.pool_prob = pool_prob
        self.fc_prob = fc_prob
        self.max_conv_kernel = max_conv_kernel
        self.max_out_ch = max_out_ch

        self.max_fc_neurons = max_fc_neurons
        self.output_dim = output_dim

        self.layers = []
        self.acc = None
        self.vel = [] # Initial velocity
        self.pBest = []

        # Build particle architecture
        self.initialization()
```

Listing A.2: Initialization Particle

```python
def update(self):
        new_p = updateParticle(self.layers, self.vel)
        new_p = self.validate(new_p)

        self.layers = new_p
        self.model = None
```

Listing A.3: Update Particle

```python
def velocity(self, gBest, Cg):
        self.vel = computeVelocity(gBest, self.pBest.layers, self.layers, Cg)
```

Listing A.4: Compute Velocity

```python

def model_compile(self, dropout_rate):
        list_layers = self.layers
        self.model = Sequential()
```

Listing A.5: Compile model

```python

def model_fit(self,train, batch_size, epochs):
        hist = self.model.fit(train_generator, validation_split=0.0, batch_size=None, epochs=epochs)

        return hist

    def model_fit_complete(self,train_generator, batch_size, epochs):
        hist = self.model.fit(train_generator, validation_split=0.0, batch_size=None, epochs=epochs)

        return hist
```

Listing A.6: Fit the model

```python

def add_conv(layers, max_out_ch, conv_kernel):
    out_channel = np.random.randint(3, max_out_ch)
    conv_kernel = np.random.randint(3, conv_kernel)

    layers.append({"type": "conv", "ou_c": out_channel, "kernel": conv_kernel})

    return layers
```

Listing A.7: Add convolution

```python
class Population:
def __init__(self, pop_size, min_layer, max_layer, input_width, input_height,
   input_channels, conv_prob, pool_prob, fc_prob, max_conv_kernel, max_out_ch,
   max_fc_neurons, output_dim):
        # Compute maximum number of pooling layers for any given particle
        max_pool_layers = 0
        in_w = input_width

        while in_w > 4:
            max_pool_layers += 1
            in_w = in_w/2

        self.particle = []
        for i in range(pop_size):
            self.particle.append(Particle(min_layer, max_layer, max_pool_layers,
   input_width, input_height, input_channels, conv_prob, pool_prob, fc_prob,
   max_conv_kernel, max_out_ch, max_fc_neurons, output_dim))
```

Listing A.8: Population class

```python
class psoCNN:
    def __init__(self, dataset, n_iter, pop_size, batch_size, epochs, min_layer,
   max_layer, \
        conv_prob, pool_prob, fc_prob, max_conv_kernel, max_out_ch,
   max_fc_neurons, dropout_rate):

        self.pop_size = pop_size
        self.n_iter = n_iter
        self.epochs = epochs

        self.batch_size = None
        self.gBest_acc = np.zeros(n_iter)
        self.gBest_test_acc = np.zeros(n_iter)
 def fit(self, Cg, dropout_rate)
 def fit_gBest(self, batch_size, epochs, dropout_rate)
 def evaluate_gBest(self, batch_size)
```

Listing A.9: PSOcnn class

```python


######## Algorithm parameters ##################
```

```



    number_runs = 10
    number_iterations = 10
    population_size = 20

    batch_size_pso = 32
    batch_size_full_training = 32

    epochs_pso = 1
    epochs_full_training = 100

    max_conv_output_channels = 256
    max_fully_connected_neurons = 300

    min_layer = 3
    max_layer = 20
    for i in range(number_runs):
        print("Run number: " + str(i))
        start_time = time.time()
        pso = psoCNN(dataset=dataset, n_iter=number_iterations, pop_size=
   population_size,
                    batch_size=batch_size_pso, epochs=epochs_pso, min_layer=
   min_layer, max_layer=max_layer,
                    conv_prob=probability_convolution, pool_prob=
   probability_pooling,
                    fc_prob=probability_fully_connected, max_conv_kernel=
   max_conv_kernel_size,
                    max_out_ch=max_conv_output_channels, max_fc_neurons=
   max_fully_connected_neurons,
                    dropout_rate=dropout)

        pso.fit(Cg=Cg, dropout_rate=dropout)
        print(pso.gBest_acc)

        # Plot current gBest
        matplotlib.use('Agg')
        plt.plot(pso.gBest_acc)
        plt.xlabel("Iteration")
        plt.ylabel("gBest acc")
        plt.savefig(results_path + "gBest-iter-" + str(i) + ".png")
        plt.close()

        print('gBest architecture: ')
        print(pso.gBest)
```

Listing A.10: Main method

Bibliography

[1] Daniel Molina, Javier Poyatos, Javier Del Ser, Salvador García, Amir Hussain, and Francisco Herrera. Comprehensive taxonomies of nature-and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis recommendations. *Cognitive Computation*, 12(5):897–939, 2020.

[2] Prabhu. Understanding of convolutional neural network (cnn) — deep learning. https://tinyurl.com/43s3ksyj, March 2018.

[3] Rohith Gandhi. Support vector machine — introduction to machine learning algorithms. https://tinyurl.com/3ny43ytm, June 2018.

[4] Grace Zhang. What is the kernel trick? why is it important? https://tinyurl.com/zsvtyasd, November 2018.

[5] Tazmina Sharmin, Fabio Di Troia, Katerina Potika, and Mark Stamp. Convolutional neural networks for image spam detection. *Information Security Journal: A Global Perspective*, 29(3): 103–117, 2020.

[6] JUSTIN LIANG. Canny edge detection. http://justin-liang.com/tutorials/canny/.

[7] Sofiane Sahir. Canny edge detection step by step in python — computer vision. https://youtu.be/FCUBwP-JTsA, January 2019.

[8] Axel Thevenot. Particle swarm optimization (pso) visually explained. https://towardsdatascience.com/particle-swarm-optimization-visually-explained-46289eeb2e14, December 2020.

[9] Jonathan Fregoso, Claudia I Gonzalez, and Gabriela E Martinez. Optimization of convolutional neural networks architectures using pso for sign language recognition. *Axioms*, 10(3):139, 2021.

[10] What is a weighted overlay? http://www.gitta.info/Suitability/en/html/WeightOverla_learningObject1.html, 2013.

[11] X-S Yang and João Paulo Papa. Bio-inspired computation and its applications in image processing: an overview. *Bio-inspired computation and applications in image processing*, pages 1–24, 2016.

[12] Kathy Lee, Diana Palsetia, Ramanathan Narayanan, Md Mostofa Ali Patwary, Ankit Agrawal, and Alok Choudhary. Twitter trending topic classification. In *2011 IEEE 11th International Conference on Data Mining Workshops*, pages 251–258. IEEE, 2011.

[13] savyakhosla. Cnn | introduction to pooling layer. https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/, July 2021.

[14] Sumit Saha. A comprehensive guide to convolutional neural networks — the eli5 way. https://tinyurl.com/494b3dsx, December 2018.

[15] Kaivan Kamali. Deep learning (part 3) - convolutional neural networks (cnn). https://tinyurl.com/3mufx5uy, May 2021.

[16] Jinde Shubham. Support vector machines (svm). https://medium.com/coinmonks/support-vector-machines-svm-b2b433419d73, June 2018.

[17] Rushikesh Pupale. Support vector machines(svm) — an overview. https://tinyurl.com/5cjhny7r, June 2018.

[18] Tejumade Afonja. Kernel functions. https://towardsdatascience.com/kernel-function-6f1d2be6091, January 2017.

[19] Czako Zoltan. Svm and kernel svm. https://tinyurl.com/tphum33r, November 2018.

[20] Prateek Bajaj. Creating linear kernel svm in python. https://www.geeksforgeeks.org/creating-linear-kernel-svm-in-python/, June 2018.

[21] Andrew Ng. Support vector machines | using an svm. https://youtu.be/FCUBwP-JTsA, January 2017.

[22] Prof. Essa. Standard deviation: Definition, examples. https://www.statisticshowto.com/probability-and-statistics/standard-deviation/, 2021.

[23] Till Buchacher. Hyperparameter optimisation utilising a particle swarm approach. https://tinyurl.com/u2ehxh55, August 2018.

[24] A.S. Bhandare. *Bio-inspired Algorithms for Evolving the Architecture of Convolutional Neural Networks*. University of Toledo, 2017. URL https://books.google.dz/books?id=xg8ZyAEACAAJ.

[25] Abdallah Anes BESTAOUI. Gestion de spectre dans un réseau de radio cognitive en utilisant l'algorithme d'optimisation par essaim de particules. http://dspace.univ-tlemcen.dz/handle/112/8104, October 2015.

[26] Achouri souhila. Etude de l'identification d'un système d'ordre fractionnaire. http://archives.univ-biskra.dz/bitstream/123456789/1, 2020.

[27] Maurice Clerc and James Kennedy. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *IEEE transactions on Evolutionary Computation*, 6(1): 58–73, 2002.

[28] Yan Gao. Image spam hunter, acoustics, speech and signal processing. 2008.

[29] Xiaonan Zhao Yan Gao, Ming Yang. Image spam hunter. https://users.cs.northwestern.edu/~yga751/ML/ISH.htm.

[30] Mark Dredze, Reuven Gevaryahu, and Ari Elias-Bachrach. Learning fast classifiers for image spam. In *CEAS*, pages 2007–487, 2007.

[31] Ari Elias-Bachrach Mark Dredze, Reuven Gevaryahu. Image spam dataset. https://www.cs.jhu.edu/~mdredze/datasets/image_spam/, 2007.

[32] Annapurna Annadatha and Mark Stamp. Image spam analysis and detection. *Journal of Computer Virology and Hacking Techniques*, 14(1):39–52, 2018.

[33] spam image dataset. https://www.dropbox.com/s/7zh7r9dopuh554e/New_Spam.zip?dl=0, .

[34] Aneri Chavda, Katerina Potika, Fabio Di Troia, and Mark Stamp. Support vector machines for image spam analysis. 2018.

[35] spam image downloadable dataset. https://users.cs.northwestern.edu/~yga751/ML/ISH.htm#dataset, .