ALGERIAN REPUBLIC DEMOCRATIC AND POPULAR

MINISTRY OF HIGH EDUCATION AND SCIENTIFIC RESEARCHES

University of Mohamed Khider Biskra Faculty of Exact Science and Science of Life and Nature Department of Computer Science

Dissertation For Master Degree Graduation In Computer Science Field

Artificial Intelligence

Training And Optimizing Deep Artificial Neural Networks Using Dragonfly Algorithm for Medical Prediction

---

Realized by:

Nedjai Azzeddine

Supervised by:

DR.Slatnia Sihem

Defended between 20-24 September 2020, in front of the jury composed of:

Dedication

This work is dedicated to:

My beloved Parents who always supported and belived in me.

My brother and sisters Hani, Asma and Imen for their encouragements.

Last but not least, to the boys with no particular order: Aziz, Red, Salouma, Cherif, Sofian, Babbi, and Ahmed.

And to all those who I love and cherich.

Acknowledgements

First and Foremost I would like to show gratitude and praise to God almighty for giving me the strength, knowledge, ability and opportunity to undertake this challenge and to persevere and complete it satisfactorily.

wish to express my sincere appreciation Dr.Slatnia Sihem for her valuable guidance.

I also wish to thank the jury members for their efforts to evaluate this work.

And finally I wish to express my deepest gratitude to to my classmates and the academic crew of the department for their assistance provided through my academic career.

Abstract

Artificial intelligence applications can be highly beneficial for healthcare in general to the extent that it can revolutionise the entire healthcare system. Artificial Neural Network (ANN) is one of the evolutionary computation techniques that can be used as a medical prediction model for new data records from shallow ANN such as Multilayer prceptron (MLP) to deep ANN such as Conversational neural networks (CNN), The ANN will be trained with an optimization method called Dragonfly Algorithm (DA) as a main focus in a model called (ANN-DA), and other well known optimizers such as Grey Wolf Optimization (GWO) and many others as an evaluation method. In order to evaluate ANN-DA. An experimental comparison has been made using different metrics. The results show that ANN-DA proved its efficiency of performance over other Models.

Keywords: Medical prediction, Artificial Neural Network (ANN), Conversational neural networks (CNN),Multilayer prceptron (MLP), Deep Learning, Dragonfly algorithm (DA), Grey Wolf Optimization (GWO), ANN-DA, ANN-GWO ,CNN-DA, MLP-DA, MLP-GWO.

Contents

List of Figures

List of Tables

General Introduction

Research in medical fields is very relevant to clinical advances . In this context, computers are changing the healthcare industry, as well as research from many perspectives, Within this scenario, machine learning, pattern recognition, and, more generally, Artificial intelligence (IA) play a very crucial role . AI is growing rapidly, and its successful application in the eHealth domain is possibly due, in general, to the availability of massive datasets and computing resources. AI has found application in many medical branches. In general, a major topic of AI in medicine is related to the clinical decision support (CDS) to assist clinicians at the point of care .[6]

Health care organizations are leveraging Deep-learning techniques, such as Convolutional neural networks (CNN),most commonly applied to analyzing visual imagery, meaning it can predict and/or diagnose diseases via Images of CT scans, X-Rays even microscopic images to improve delivery of care at a reduced cost.

In order for CNNs to work, they need to be trained, by optimization algorithms such as Adaptive Moment Estimation (ADAM), Stochastic gradient descent(SGDM)... CNNs can be also trained by nature based optimization algorithms, it is rareley studied, and that is why it's going to be our main focus in this project

In this project we are going to focus on training artficial neural networks from simple multilayer perceptrons(MLP) to deep CNNs using a novel nature based optimization technique called Dragonfly algorithm DA, And as a final goal we going to build a CNN model trained by DA called CNN-DA for COVID-19 diagnosis.

We decided to divide this thesis into a general introduction, a general conclusion, and 4 chapters where we talk about machine learning and its different techniques and methods in the 1st chapter, in the 2nd chapter we will dive into meta-heuristics and optimization algorithms where we discuss the optimization techniques used in our project, the 3rd chapter will be the design of our project where we design a solution for our research, finally the 4th chapter will be the implementation of our new architectures and the test results obtained .

Chapter 1

Machine Learning

1.1 Iintroduction

We have seen Machine Learning as a buzzword for the past few years, the reason for this might be the high amount of data production by applications, the increase of computation power in the past few years and the development of better algorithms.

Machine Learning is used anywhere from automating mundane tasks to offering intelligent insights, industries in every sector try to benefit from it. But there are much more examples of ML in use.

• Image recognition — Machine learning can be used for face detection in an image as well. There is a separate category for each person in a database of several people.

• Speech Recognition — It is the translation of spoken words into the text. It is used in voice searches and more. Voice user interfaces include voice dialing, call routing, and appliance control. It can also be used a simple data entry and the preparation of structured documents.

• Financial industry and trading — companies use ML in fraud investigations and credit checks.

• Prediction and Medical diagnoses — ML is trained to recognize and predict whether a person is sick or not, which will be our main focus in this project.

1.2 What is Machine Learning ?

According to Arthur Samuel, Machine Learning algorithms enable the computers to learn from data, and even improve themselves, without being explicitly programmed.

Machine learning (ML) is a category of an algorithm that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that can receive input data and use statistical analysis to predict an output while updating outputs as new data becomes available.[15]

1.3 Types of Machine Learning

Machine learning can be classified into 3 types of algorithms.

1. Supervised Learning.
2. Unsupervised Learning.
3. Reinforcement Learning.[15]

1.3.1 Overview of Supervised Learning

In Supervised learning, a system is presented with data which is labeled, which means that each data tagged with the correct label.

The goal is to approximate the mapping function so well that when you have new input data (x) that you can predict the output variables (Y) for that data.[15]

In this project this is what we will be working with.

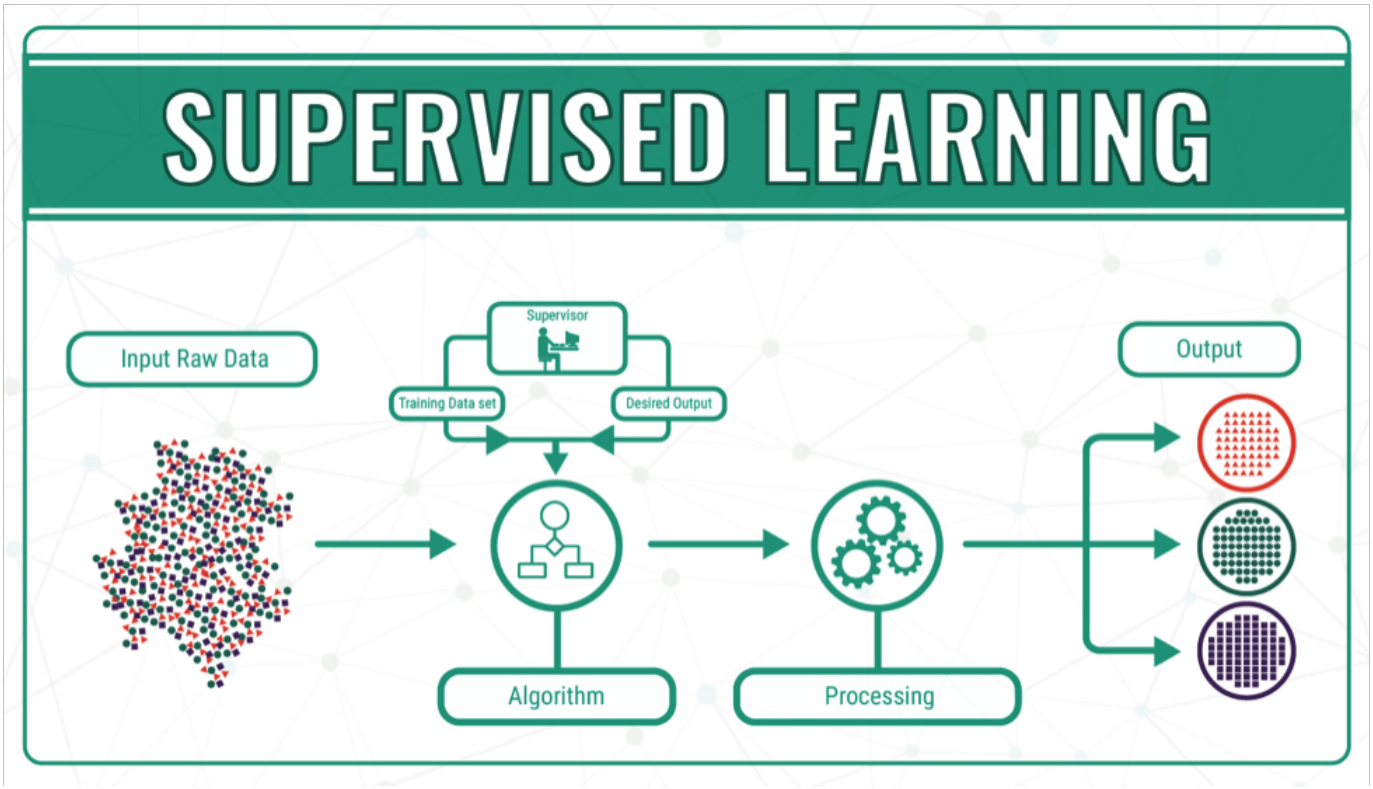

Figure 1.1: Supervised Learning[22]

Types of Supervised Learning

•**Regression:** A regression problem is when the output variable is a real value, such as "dollars" or "weight"

•**Classification:** A classification problem is when the output variable is a category, such as "red" or "blue" or "disease" and "no disease". Therefore Classification is very suitable for this project if we are dealing with medical prediction and diagnosis.[15]

1.3.2 Overview of Unsupervised Learning

In unsupervised learning, a system is presented with unlabeled, uncategorized data and the system's algorithms act on the data without prior training. The output is dependent upon the coded algorithms. Subjecting a system to unsupervised learning is one way of testing AI.[15]

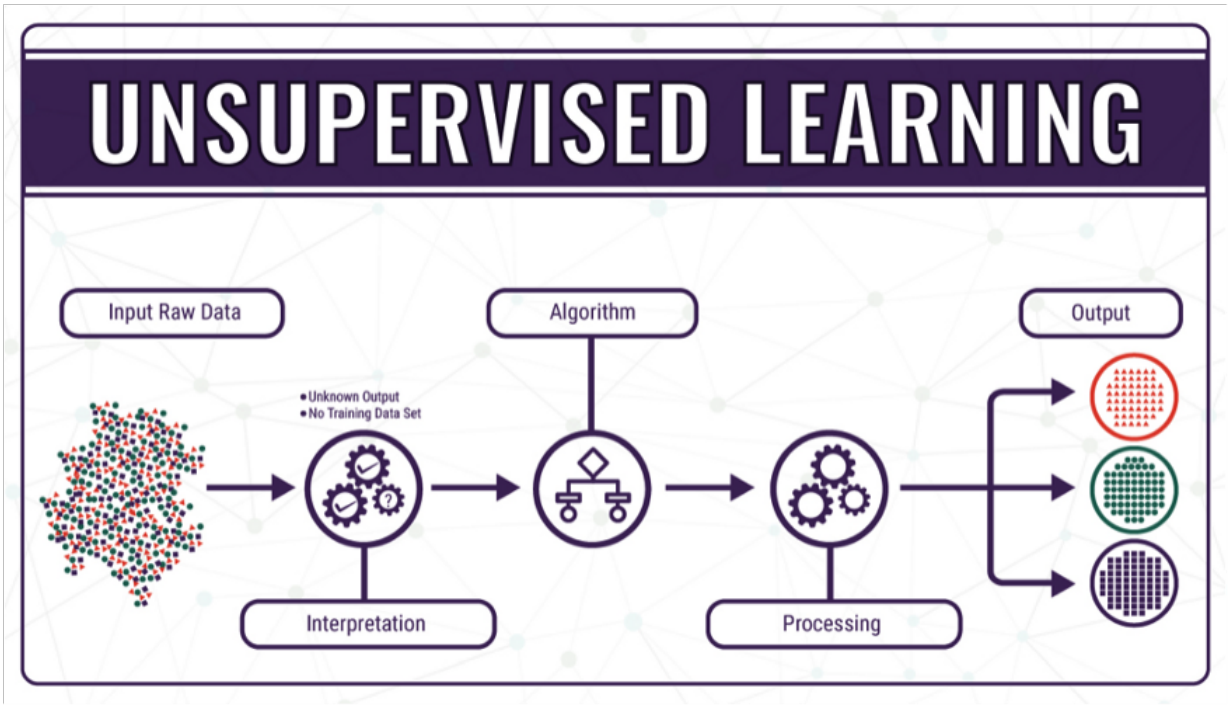

Figure 1.2: Unsupervised Learning[22]

Types of Unsupervised Learning

•**Clustering:** A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

•**Association:** An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.[15]

1.3.3 Reinforcement Learning

A reinforcement learning algorithm, or agent, learns by interacting with its environment. The agent receives rewards by performing correctly and penalties for performing incorrectly. The agent learns without intervention from a human by maximizing its reward and minimizing its penalty. It is a type of dynamic programming that trains algorithms using a system of reward and punishment.[15]

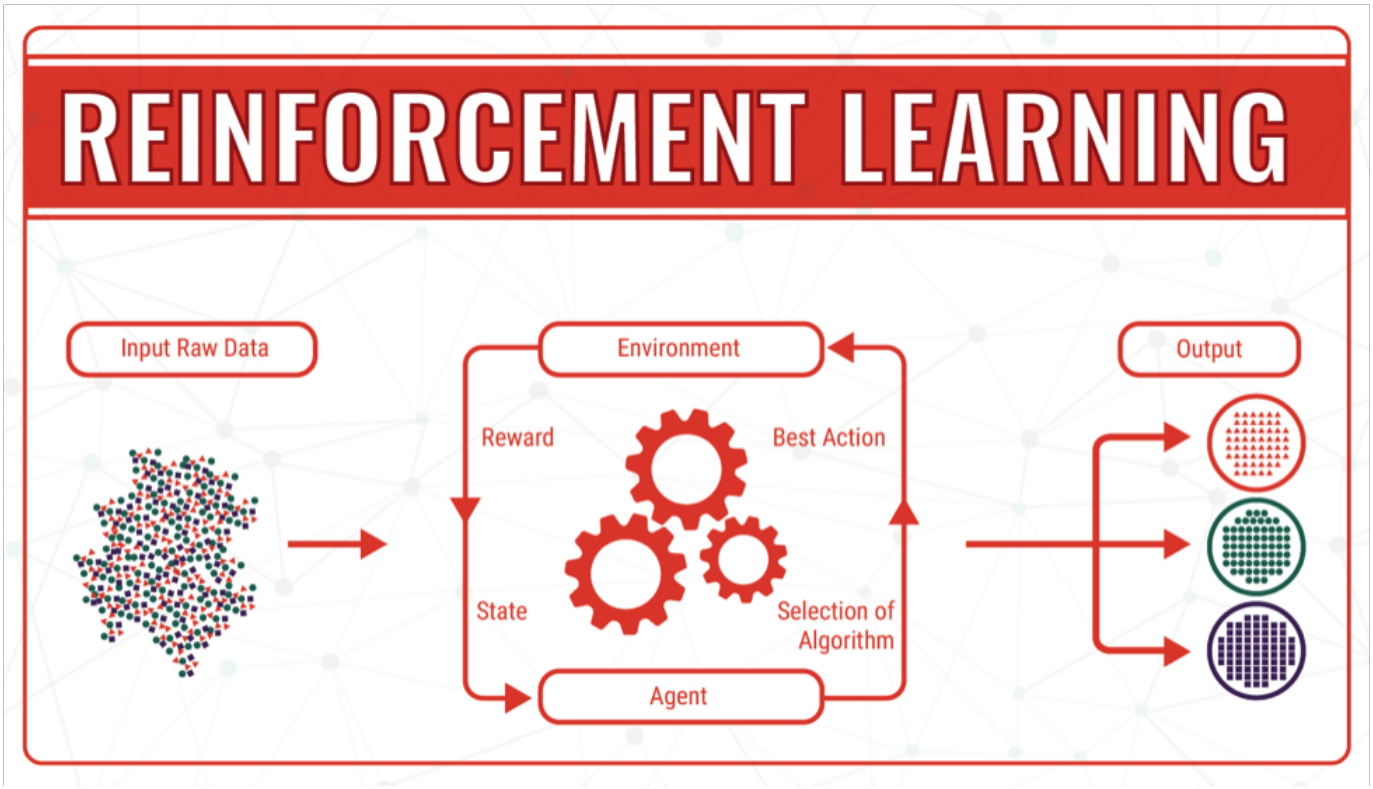

Figure 1.3: Reinforcement Learning[8]

1.4 Classification

In this project we will be using supervised learning , and because we are dealing with medical prediction and diagnosis we find that Classification is suitable for such tasks.

1.4.1 What is Classification?

Classification is the process of predicting the class of given data points. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).

Classification belongs to the category of supervised learning where the targets also provided with the input data. There are many applications in classification in many domains such as in credit approval, medical diagnosis, target marketing etc.[2]

1.4.2 Classification Algorithms

There is a lot of classification algorithms available now but it is not possible to conclude which one is superior to other. It depends on the application and nature of available data set.[2]

In this project we will be working with artificial neural network (ANN) which we will get to after we discuss some examples of Classification Algorithms .

Decision Tree

Decision tree builds classification models in the form of a tree structure. It utilizes an if-then rule set which is mutually exclusive and exhaustive for classification. The rules are learned sequentially using the training data one at a time. Each time a rule is learned, the tuples covered by the rules are removed. This process is continued on the training set until meeting a termination condition.

The tree is constructed in a top-down recursive divide-and-conquer manner. All the attributes should be categorical. Otherwise, they should be discretized in advance. Attributes in the top of the tree have more impact towards in the classification and they are identified using the information gain concept.

A decision tree can be easily over-fitted generating too many branches and may reflect anomalies due to noise or outliers. An over-fitted model has a very poor performance on the unseen data even though it gives an impressive performance on training data. This can be avoided by pre-pruning which halts tree construction early or post-pruning which removes branches from the fully grown tree.[2]

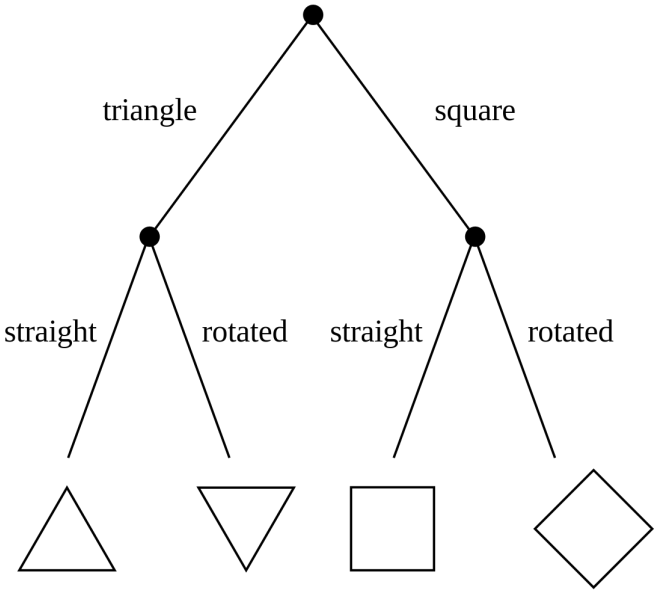

Figure 1.4: Decision Tree[2]

Naive Bayes

Naive Bayes is a probabilistic classifier inspired by the Bayes theorem under a simple assumption which is the attributes are conditionally independent.

$$P(\mathbf{X} \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i) = P(x_1 \mid C_i) \times P(x_2 \mid C_i) \times \ldots \times P(x_n \mid C_i)$$

The classification is conducted by deriving the maximum posterior which is the maximal P(Ci—X) with the above assumption applying to Bayes theorem. This assumption greatly reduces the computational cost by only counting the class distribution. Even though the assumption is not valid in most cases since the attributes are dependent, surprisingly Naive Bayes has able to perform impressively.

Naive Bayes is a very simple algorithm to implement and good results have obtained in most cases. It can be easily scalable to larger datasets since it takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Naive Bayes can suffer from a problem called the zero probability problem. When the conditional probability is zero for a particular attribute, it fails to give a valid prediction. This needs to be fixed explicitly using a Laplacian estimator.

K-Nearest Neighbour

k-Nearest Neighbor is a lazy learning algorithm which stores all instances correspond to training data points in n-dimensional space. When an unknown discrete data is received, it analyzes the closest k number of instances saved (nearest neighbors)and returns the most common class as the prediction and for real-valued data it returns the mean of k nearest neighbors.

In the distance-weighted nearest neighbor algorithm, it weights the contribution of each of the k neighbors according to their distance using the following query giving greater weight to the closest neighbors.

Distance calculating query

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

Usually KNN is robust to noisy data since it is averaging the k-nearest neighbors.[2]

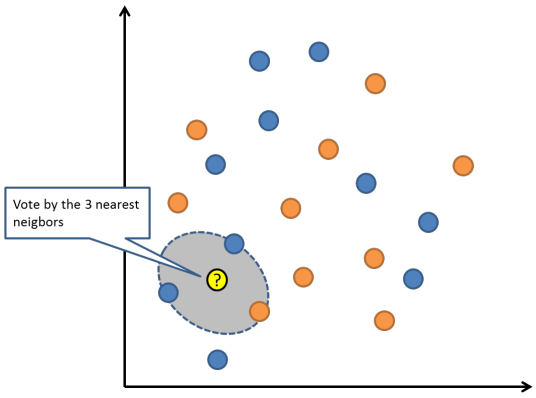

Figure 1.5: K-Nearest neighbour KNN[2]

Artificial Neural Networks ANN

A definition of Artificial Neural Network (ANN) is : a computing system made up of a number of simple, highly interconnected processing units, which process information by their dynamic state response to external inputs. where each connection has a weight associated with it .

During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples.[2]

1.5 ANN Architectures

There are many network architectures available now like :

1.5.1 Feed-forward Neural Network FNN

FNNs are those NNs with only one-way and one-directional connections between their neurons. In this type of NNs, neurons are arranged indifferent parallel layers . The first layer is always called the input layer, whereas the last layer is called the output layer. Other layers between the input and output layers are called hidden layers .[11]

Multilayer Perceptron MLP

A multi layer perceptron (MLP) is a class of feed forward artificial neural network. MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training.[12]

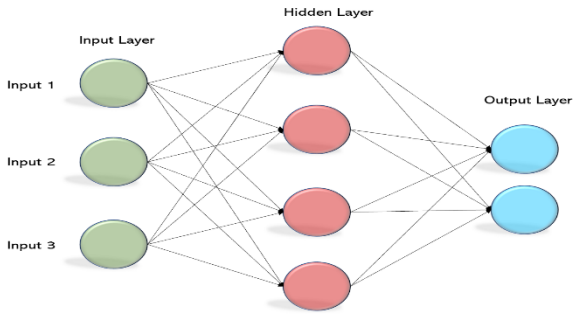

Figure 1.6: Multilayer Perceptron MLP[12]

After providing the inputs, weights, and biases, the output of our ANN are calculated throughout the following steps[11]: • Step A : The weighted sums of inputs are first calculated by:

$$s_j = \sum_{i=1}^{n} (W_{ij}.X_i) - \theta_j, \quad j = 1, 2, \ldots h$$

where n is the number of the input nodes, Wij shows the connection weight from the ith

node in the input layer to the jth node in the hidden layer, Θj is the bias (threshold) of the jth hidden node, and Xi indicates the ith input.

• Step B : The output of each hidden node is calculated as follows:

$$S_j = sigmoid\left(s_j\right) = \frac{1}{\left(1 + \exp\left(-s_j\right)\right)}, j = 1, 2, \ldots h$$

• Step C : The final outputs are defined based on the calculated outputs of the hidden nodes as follows:

$$o_k = \sum_{j=1}^{h} \left(w_{jk}.S_j\right) - \theta_k^{'}, k = 1, 2, \ldots, m$$

$$O_k = sigmoid\left(o_k\right) = \frac{1}{\left(1 + \exp\left(-o_k\right)\right)}, k = 1, 2, \ldots, m$$

where wjk is the connection weight from the j th hidden node to the kth output node, and Θ'k is the bias (threshold) of the kth output node.

As may be seen in the stepA to stepC, the weights and biases are responsible for defining the final output of MLPs from given inputs. Finding proper values for weights and biases in order to achieve a desirable relation between the inputs and outputs is the exact definition of training MLPs.[11]

1.5.2 Recurrent Neural Network

A Recurrent neural network is distinguished from an anticipatory neural network in that it has at least one feedback loop . For example, a recurrent network may consist of a single layer of neurons with each neuron supplying its output signal to the inputs of all other neurons[3]

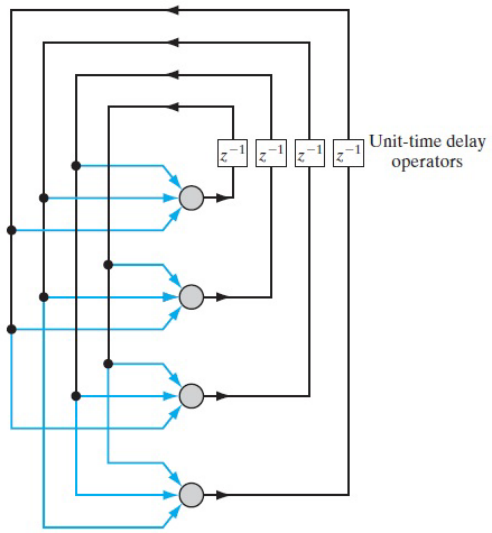

Figure 1.7: Recurrent network with no self-feedback loops and no hidden neurons[3]

We illustrate another class of recurrent networks with hidden neurons. The illustrated feedback connections also come from hidden neurons. from the output neurons.

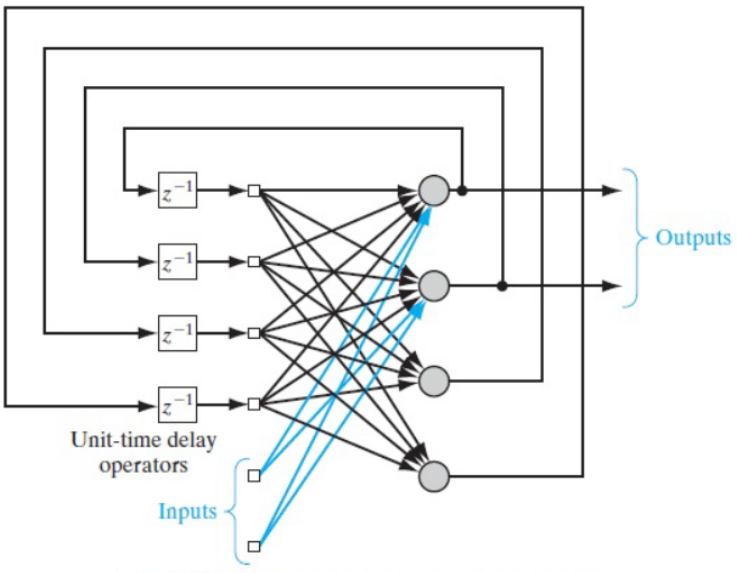

Figure 1.8: Recurrent network with hidden neurons[3]

1.5.3 Convolutional Neural Networks CNN

CNN is another Feed-forward neural network that we will be focusing on in this project, CNNs is a type of acyclic artificial neural network (feed-forward), in which the connection pattern between neurons is inspired by the animal's visual cortex.

Neurons in this region ofthe brain are arranged so that they correspond to overlapping regions when paving the visualfield .

The goal of a CNN is to learn higher-order features in the data via convolutions. They are wellsuited to object recognition with images and consistently top image classification competitions.They can identify faces, individuals, street signs, platypuses, and many other aspects of visualdata. CNNs overlap with text analysis via optical character recognition, but they are also useful when analyzing words as discrete textual units. They're also good at analyzing sound, and at building position and (somewhat) rotation invariant features from raw image data.[3]

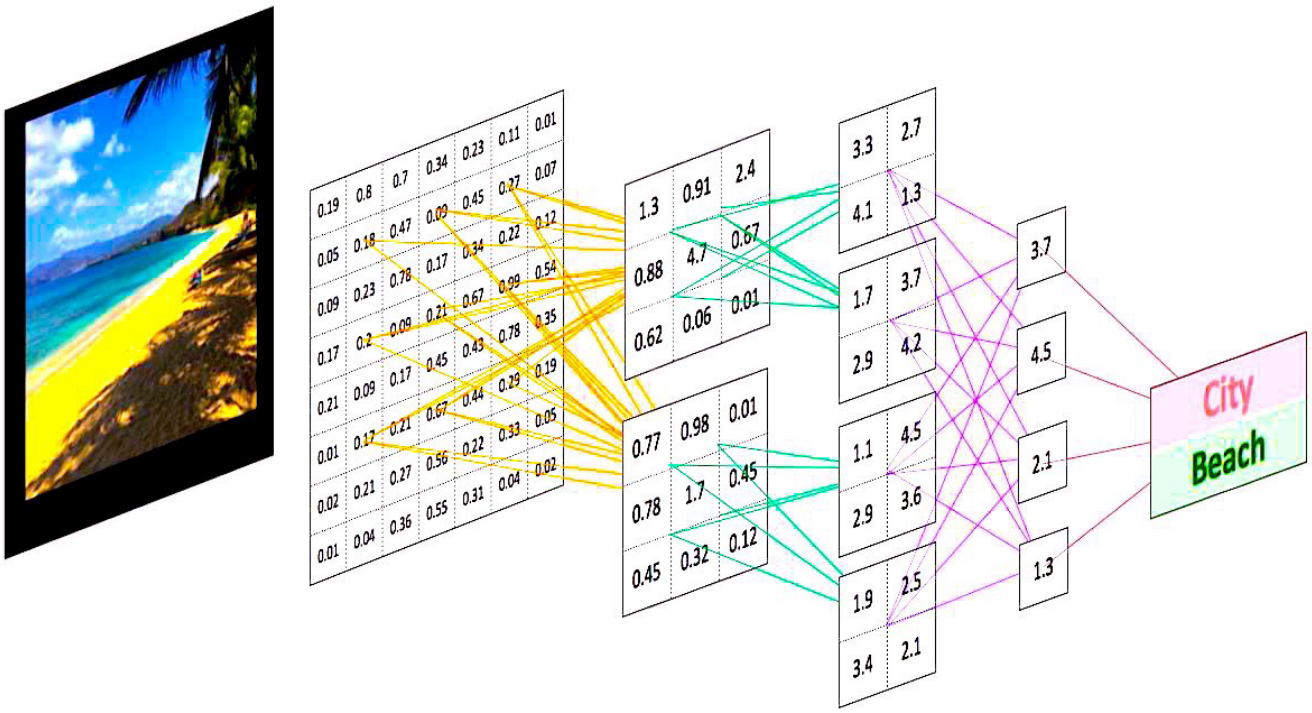

Figure 1.9: Convolutional Neural Network CNN[3]

1.6 Convolutional Neural network CNN

Before getting into Convolutional Neural networks, we need to talk about Deep Learning

1.6.1 Deep Learning

(also known as deep structured learning, hierarchical learning or deep machine learning) is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data.

In a simple case, you could have two sets of neurons: ones that receive an input signal and ones that send an output signal. When the input layer receives an input it passes on a modified version of the input to the next layer. In a deep network, there are many layers between the input and output (and the layers are not made of neurons but it can help to think of it that way), allowing the algorithm to use multiple processing layers, composed of multiple linear and non-linear transformations.

Deep Learning has revolutionized the machine learning recently with some of the great works being done in this field. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics.

But, the ancient term “Deep Learning” was first introduced to Machine Learning by Dechter (1986), and to Artificial Neural Networks (NNs) by Aizenberg et al (2000).

It was further popularized by the development of Convolutional Networks Architecture by Alex Krizhevsky named ‘AlexNet’ that won the competition of ImageNet in 2012 by defeating all the image processing methods and creating a way for deep learning architectures to be used in Image Processing.[21]

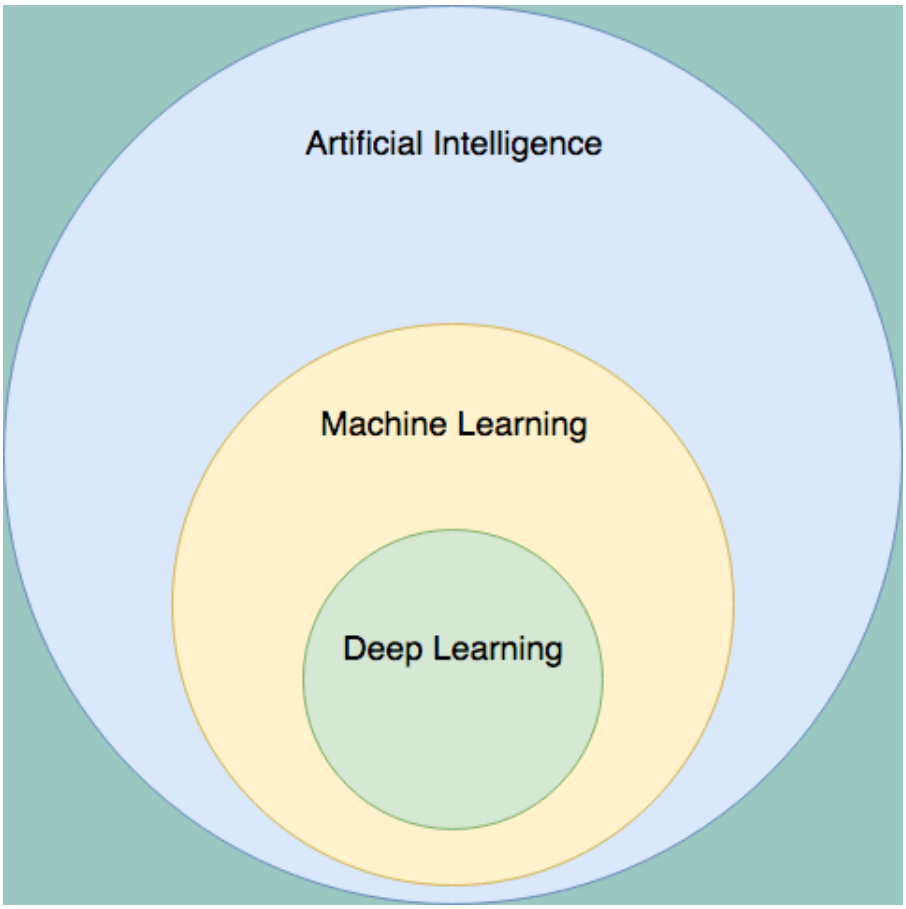

Figure 1.10: Deep Learning is a subset of Machine Learning and Machine learning is a subset of IA. [4]

1.6.2 Convolutional Neural network CNN

In Deep Learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.

Individual cortical neurons respond to stimuli in a restricted region of space known as the receptive field. The receptive fields of different neurons partially overlap such that they tile the visual field.

The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation. Convolutional networks were inspired by biological processes and are variations of multilayer perceptrons designed to use minimal amounts of preprocessing.

They have wide applications in image and video recognition, recommender systems and natural language processing.

LeNet was one of the very first convolutional neural networks which helped propel the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since the year 1988.

At that time the LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits, etc.[21]

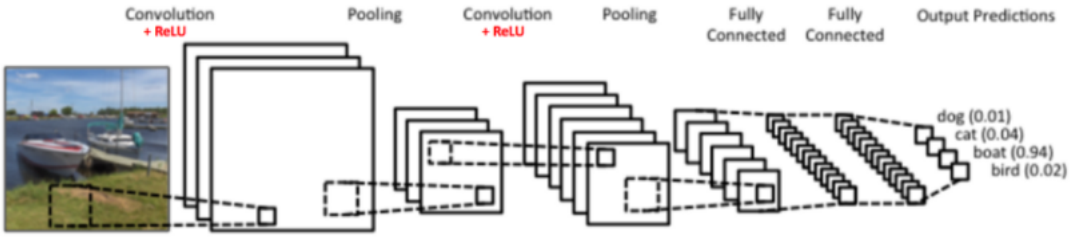

Figure 1.11: A simple CNN model [21]

There are four main components in a ConvNet : 1. Convolutional Layer 2. Activation Function 3. Pooling Layer 4. Fully Connected Layer

Convolutional layer

Convolutional Layer is based on the term 'Convolution', which is a mathematical operation performed on two variables (f*g) to produce a third variable. It is similar to cross-correlation. The input to a convolutional layer is a m x m x r image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3. The convolutional layer will have k filters (or kernels) of size n x n x q where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size mn+1.8 [21]

Activation Function

To implement complex mapping functions, activation functions are needed, that are non-linear in order to bring in the much needed non-linearity property that enables them to approximate any function. Activation functions are also important for squashing the unbounded linearly weighted sum from neurons.

This is important to avoid large values accumulating high up the processing hierarchy. A lot of activation functions are present that can be used with some of the primarily used ones being sigmoid, tanh and ReLU.[21]

Pooling Layer

Pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing it's dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.

This is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation invariance to the internal representation.

Some of the most prominently used pooling techniques are Max-Pooling, MinPooling and Average-Pooling.[21]

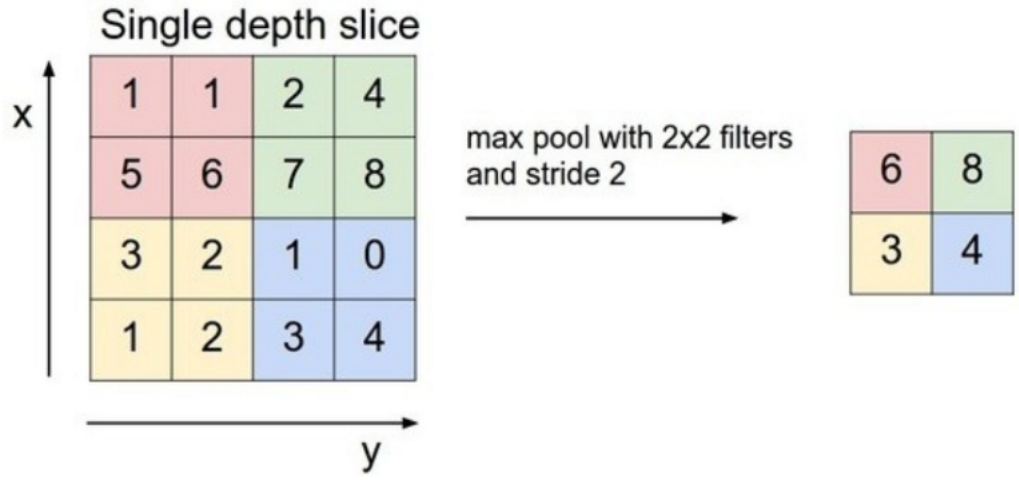

Figure 1.12: Example of Max-Pooling of 2*2 filters[21]

Fully Connected Layer

The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer. The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function or any other similar function in the output layer. [21]

1.6.3 Residual Neural Network ResNet

ResNets or Residual Networks are a type of Convolutional Neural Network (CNN) architecture introduced by Kaiming He in his paper 'Deep Residual Learning for Image Recognition' in 2015. This architecture shows a way to train networks with as many as 1000 layers. This helped him win the ILSVRC 2015 classification task.

Previously, it was believed that it is not possible to train models with more than 20 layers. The best CNN architecture before ResNets was the VGG19, as the name suggests this model has 19 convolutional layers. And it wasn't possible to add more layers to this architecture because when adding more layers we would encounter a problem commonly referred to as the 'Vanishing Gradient' problem. This basically means that if you have more than 8 layers in your architecture you would reach a point during training when gradients would become infinitely large or become zero hence making it impossible to train the model. ResNet provides us with a way to overcome this problem and create models which can have as many as 1000 layers, thus creating models which can represent more complex features.

The answer that ResNet provided us with is Skip Connections. This means that suppose we consider two convolutional layers as a single block which we will call the Res block then the input to this ResBlock has another path which connects it to the output of the Res block. These inputs are then added to the output of the Res block giving us the final output for that particular ResBlock. This ensures that during training even if we encounter the vanishing gradient problem and all the weights in the Res block become zero the output of the Res block is not zero because the final output of the ResBlock is the output plus input of the Res block, thus ensuring that output of a Res block can never be zero. This ensures that models can be trained with a theoretically infinite number of layers provided that all of these layers are divided into Res blocks with Skip Connections which map the input to the Res block to the

output.[20]

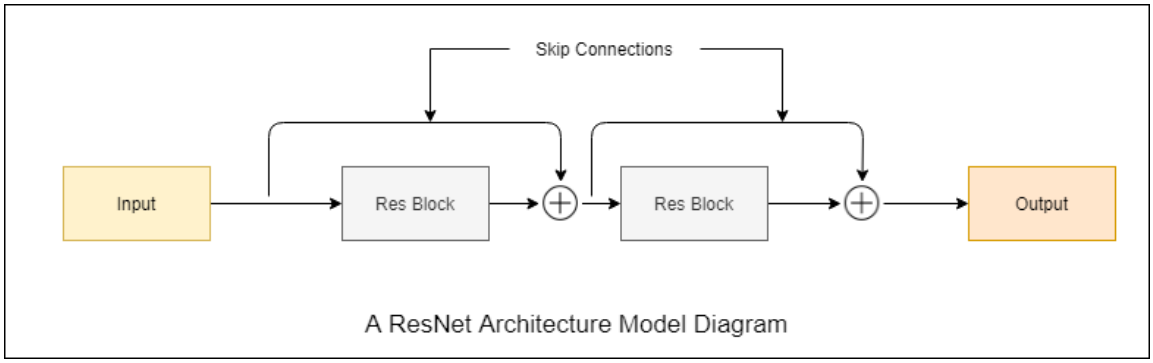

Figure 1.13: Resnet Architecture model diagram[20]

The image above shows a 5 layer ResNet Model. 2 layers each inside the Res blocks and 1 output layer. Here, the input is connected to the first ResBlock and also to the output of the first ResBlock where it gets added to the output of the first ResBlock. Then, this output is considered to be the input to the next ResBlock where again this input has two paths. In this way we can have N number of Res blocks. This is the layout of a ResNet Architecture model. The Res blocks typically have 2 convolutional layers in them but we can have as many as we want.

ResNet is one of the most powerful deep neural networks which has achieved fantabulous performance results in the ILSVRC 2015 classification challenge. ResNet has achieved excellent generalization performance on other recognition tasks and won the first place on ImageNet detection, ImageNet localization, COCO detection and COCO segmentation in ILSVRC and COCO 2015 competitions. There are many variants of ResNet architecture i.e. same concept but with a different number of layers. We have ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-110, ResNet-152, ResNet-164, ResNet-1202 etc. The name ResNet followed by a two or more digit number simply implies the ResNet architecture with a certain number of neural network layers[5]

ResNet-50

ResNet-50 is a pretrained Deep Learning model for image classification of the Convolutional Neural Network(CNN, or ConvNet), which is a class of deep neural networks, most commonly applied to analyzing visual imagery. ResNet-50 is 50 layers deep and is trained on a million

images of 1000 categories from the ImageNet database. Furthermore the model has over 23 million trainable parameters, which indicates a deep architecture that makes it better for image recognition. Using a pretrained model is a highly effective approach, compared if you need to build it from scratch, where you need to collect great amounts of data and train it yourself. Of course, there are other pretrained deep models to use such as AlexNet, GoogleNet or VGG19, but the ResNet-50 is noted for excellent generalization performance with fewer error rates on recognition tasks and is therefore a useful tool to know.[17]

1.6.4 Resnet-50 Architecure

Now we'll talk about the architecture of ResNet50. The architecture of ResNet50 has 4 stages as shown in the diagram below. The network can take the input image having height, width as multiples of 32 and 3 as channel width. For the sake of explanation, we will consider the input size as 224 x 224 x 3. Every ResNet architecture performs the initial convolution and max-pooling using 77 and 33 kernel sizes respectively. Afterward, Stage 1 of the network starts and it has 3 Residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 128 respectively. The curved arrows refer to the identity connection. The dashed connected arrow represents that the convolution operation in the Residual Block is performed with stride 2, hence, the size of input will be reduced to half in terms of height and width but the channel width will be doubled. As we progress from one stage to another, the channel width is doubled and the size of the input is reduced to half.

For deeper networks like ResNet50, ResNet152, etc, bottleneck design is used. For each residual function F, 3 layers are stacked one over the other. The three layers are 11, 33, 11 convolutions. The 11 convolution layers are responsible for reducing and then restoring the dimensions. The 33 layer is left as a bottleneck with smaller input/output dimensions.

Finally, the network has an Average Pooling layer followed by a fully connected layer having 1000 neurons (ImageNet class output).[17]

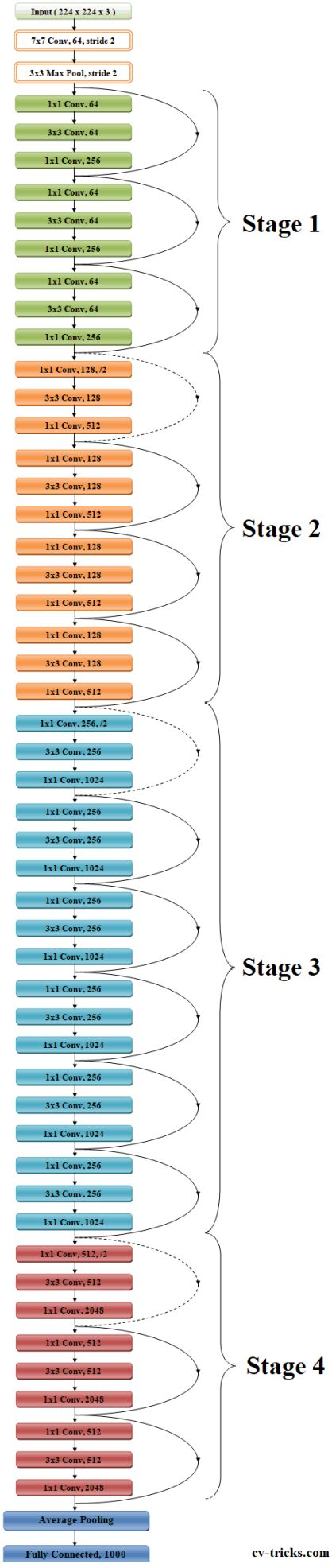

Figure 1.14: Resnet-50 Architecture model diagram[17]

1.7 Related Work (Medical Prediction and Diagnosis)

1.7.1 Prediction of Chronic Kidney DiseasesUsing Deep Artificial Neural Network Technique

The progression of the chronic kidney disease and methodologies to diagnose chronic kidney disease is a challenging problem which can reduce the cost of treatment.

In 2019 J. Dinesh, Steven Lawrence, Eduardo Thomaz, studied 224 records of chronic kidney disease available on the UCI machine learning repository named chronic kidney diseases dating back to 2015. Their proposed method is based on deep neural network which predicts the presence or absence of chronic kidney disease with an accuracy of 97%. Compared to other available algorithms, the model built shows better results which is implemented using the cross-validation technique to keep the model safe from overfitting. This automatic chronic kidney disease treatment helps reduce the kidney damage progression, but for this chronic kidney disease detection at initial stage is necessary.[7]

1.7.2 Heart disease prediction using K-means and Artificial Neural Network

The heart is an important organ of human body part. Life is completely dependent on efficient working of the heart. What if a heart undergoes a disorder, cardiovascular diseases are the most challenging disease for reducing patient count. According to survey conducted by WHO, about 17 million people die around the globe due to cardiovascular diseases i.e 29.20% among all caused death, mostly in developing countries. Thus there is a need of getting rid of the this complicated task CVD using advanced data machine learning techniques, in order to discover knowledge of Heart disease prediction. In 2017 Malav, Amita, Kadam, Kalyani, Kamat, Pooja proposeed an efficient hybrid algorithmic approach for heart disease prediction that serves efficient prediction technique to determine and extract the unknown knowledge of heart disease using hybrid combination of K-means clustering algorithm and artificial neural network.

This proposed model considered 14 attribute out of 74 attributes of UCI Heart Disease Data Set This technique uses medical terms such as age, weight, gender, blood pressure and

cholesterol rate etc for prediction. To perform grouping of various attributes it uses k-means algorithm and for predicting it uses Back propagation technique in neural networks. The main objective of this research is to develop a prototype for predicting heart diseases with higher accuracy rate.[1]

1.7.3 Prediction of Fatty Liver Disease using Machine Learning Algorithms

Fatty liver disease (FLD) is a common clinical complication; it is associated with high morbidity and mortality. However, an early prediction of FLD patients provides an opportunity to make an appropriate strategy for prevention, early diagnosis and treatment. In 2018 Wu, Chieh-Chen, Yeh, Wen-Chun, Hsu, and Wen-Ding aimed to develop a machine learning model to predict FLD that could assist physicians in classifying high-risk patients and make a novel diagnosis, prevent and manage FLD.

Methods

all patients who had an initial fatty liver screening at the New Taipei City Hospital between 1st and 31st December 2009 were included. Classification models such as random forest (RF), Naïve Bayes (NB), artificial neural networks (ANN), and logistic regression (LR) were developed to predict FLD. The area under the receiver operating characteristic curve (ROC) was used to evaluate performances among the four models.

Results

A total of 577 patients were included in this study; of those 377 patients had fatty liver. The area under the receiver operating characteristic (AUROC) of RF, NB, ANN, and LR with 10 fold-cross validation was 0.925, 0.888, 0.895, and 0.854 respectively. Additionally, The accuracy of RF, NB, ANN, and LR 87.48, 82.65, 81.85, and 76.96 percent.[23]

1.7.4 Building Risk Prediction Models for Type 2 Diabetes Using Machine Learning Techniques

As one of the most prevalent chronic diseases in the United States, diabetes, especially type 2 diabetes, affects the health of millions of people and puts an enormous financial burden on the US economy. In 2019 Xie, Zidian, Nikolayeva, Olga, Luo, Jiebo, and Li, Dongmei aimed to develop predictive models to identify risk factors for type 2 diabetes, which could help facilitate early diagnosis and intervention and also reduce medical costs.

Methods

cross-sectional data on 138,146 participants was analyzed, including 20,467 with type 2 diabetes, from the 2014 Behavioral Risk Factor Surveillance System. several machine learning models for predicting type 2 diabetes were built, including support vector machine, decision tree, logistic regression, random forest, neural network, and Gaussian Naive Bayes classifiers. univariable and multivariable weighted logistic regression models were used to investigate the associations of potential risk factors with type 2 diabetes.

Results

All predictive models for type 2 diabetes achieved a high area under the curve (AUC), ranging from 0.7182 to 0.7949. Although the neural network model had the highest accuracy (82.4%), specificity (90.2%), and AUC (0.7949), the decision tree model had the highest sensitivity (51.6%) for type 2 diabetes. It was found that people who slept 9 or more hours per day (adjusted odds ratio [aOR] = 1.13, 95% confidence interval [CI], 1.03–1.25) or had checkup frequency of less than 1 year (aOR = 2.31, 95% CI, 1.86–2.85) had higher risk for type 2 diabetes.[27]

1.8 Summary

In this chapter, we discussed the concepts of Machine Learning.

Machine learning, predictive analytics, and other related topics are very exciting and powerful fields. While these topics can be very technical, many of the concepts involved are relatively simple to understand at a high level. In many cases, a simple understanding is all that's required to have discussions based on machine learning problems, projects, techniques, and so on.

Based on our study of this area, we found that existing related works have certain limitations despite the benefits that offer them. **This prompted us to propose our contribution, which is in including bio-isnpired and metaheursitic appoaches and methods to optimize and train neural networks for better performance in medical prediction** .

In the next chapter we will provide an introduction to optimization and metaheuristics Algorithms as well as Nature inspierd optimization Algorithms and we will have an indepth discussion about the methods used in our project.

Chapter 2

Optimization Algorithms

2.1 Introduction

Optimization is the most essential ingredient in the recipe of machine learning algorithms. It starts with defining some kind of loss function/cost function and ends with minimizing it using one or the other optimization routine. The choice of optimization algorithm can make a difference between getting a good accuracy in hours or days. The applications of optimization are limitless and is widely researched topic in industry as well as academia.

In the this chapter we will provide an overview to optimization and metaheuristics Algorithms as well as Nature/bio inspierd optimization Algorithms and we will have an indepth discussion about the methods/techniques used in our project.

2.2 Metaheuristics

2.2.1 Definition

Meta-heuristics were conceived as high-level problem solving strategies to coordinate the cooperation between other search methods, including heuristics and/or traditional search techniques (Glover 1986). The main idea was to provide a guide for designing general problem-independent strategies, which should be later instantiated by researchers to solve specific hard problems.

This way, meta heuristics are not specifically focused on solving any kind of problems, but they propose simple ideas with high applicability to a wide number of problems. These simple procedures are usually based on emulating natural or physical phenomena, such as the behavior of flocks of birds and insects, cooling procedures in metals, or the natural evolution, among many others.

Nowadays, the classic definition of meta-heuristic has been extended in order to include a wide range of search and optimization procedures, as well as other significantly complex learning processes. The 'meta' component of a meta-heuristic refers to the higher level strategy that controls a set of underlying heuristic techniques in order to improve the search capabilities of the resulting algorithm. Meta-heuristics apply this search strategy in a neighborhood of the current solution. The search ends when a specific stopping criterion is met.

The stopping criterion is usually related to a specified number of iterations, a quality threshold for solutions or the best solution found, a time limit for the search, or any combination of the aforementioned criteria. The best solution found on the search, or a given set of best solutions, are returned. The main goals of the search when using metaheuristics can be summarised to:

1. find efficiently (i.e., quickly) feasible solutions for a given optimization problem (indeed, finding feasible solutions can be a NP-hard problem itself)

2. find efficiently (i.e., quickly) good quality solutions (with objective function values near to the optimum, or accurate enough for the problem solver).

3. cover the solution space without getting stuck in specific zones (especially local optima).

Regarding the aforementioned third goal, a specific goal of the search in a metaheuristic is to achieve an accurate and balanced pattern for diversification and intensification of solutions. These two concepts are the key for any successful search or optimization method. Diversification refers to the capability of the technique for achieving a good exploration pattern for the search space, usually providing a reasonable coverage and avoiding stagnation in local optima. On the other hand, intensification is the property of the technique that allows exploiting (i.e., improving, from the point of view of the objective function value) accurate solutions already found in the search, in order to increase their quality. Figure depicts the ideas of diversification and intensification. Depending on the method and the search strategy, metaheuristics provide

different trade-off levels between diversification and intensification. Thus, they have emerged as both efficient and robust methods for optimization.[14]

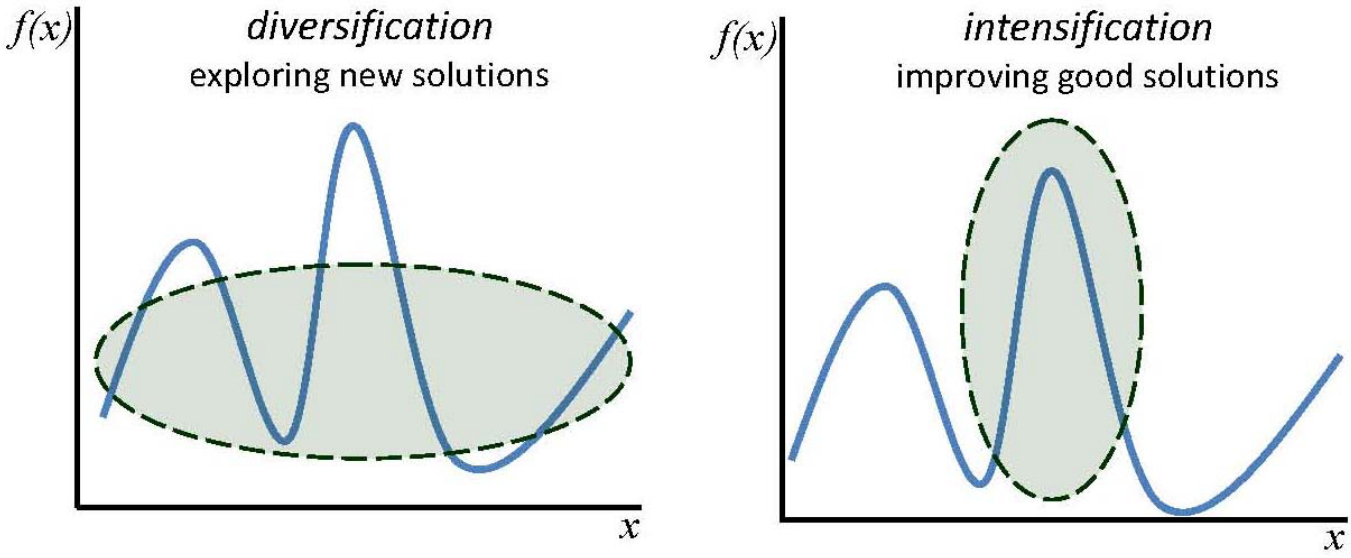

Figure 2.1: Diversification and intesification [14]

2.2.2 Metaheuristics classification criteria

• Several classification criteria have been proposed for metaheuristics. The most used classification applied in the literature takes into account the number of tentative solutions used in each step of the iterative algorithm. Other relevant classification criteria formetaheuristics are related to the procedures used within the search strategy, i.e., constructive methods, use of memory, etc.

• Regarding the number of solutions handled in each step of the iterative search, metaheuristics are classified in two categories: trajectory-based metaheuristics and population-based metaheuristics.

• **Trajectory-based metaheuristics** work with a single candidate solution, which is modified in each step, to be replaced by another (often the best) solution found in its neighbourhood, which is defined by (possibly a set of) transformation operator(s) or movement(s). In fact, this search is characterised by a trajectory in the space of candidate solutions to solve the problem. Trajectory-based metaheuristics are usually faster and more efficient than population-based metaheuristics, since they only work with a single solution at the time. Thus, they allow finding quickly a locally optimal solution and they are usually referred as exploitation-oriented

methods, which promote intensification in the search space, by locally improving already found solutions with good quality.

• On the other hand, **population-based metaheuristics** work with a set of multiple candidate solutions in each step of the search. These solutions are modified and/or combined following some common guidelines. The population paradigm allows the recombination of solutions, trying to compute better results by using the main features of the original solutions. In each iteration, some solutions in the population are replaced by newly generated solutions, often the best ones, or some selected solutions according to a quality-based criterion. As the main feature of population-based methods is to provide an increased diversification by using many candidate solutions in the population, this kind of metaheuristics are characterised as exploration-oriented methods.

• Regarding other useful classification criteria often used to characterise metaheuristics, we can distinguish between local search-based, i.e., those that use a local strategy to improve solution(s), based on searching in the neighbourhood of the current ones(s); and constructive metaheuristics, which apply a specific technique to build new solution(s), based on the characteristics of the current one(s). In addition, we can distinguish between memory-based,i.e., those metaheuristics that use a specific structure to account forprevious information about the search and memory-less, i.e., those methods that do not use memory, and the search performed in each step is independent of the previous decisions taken during the search.[14]

2.2.3 Metaheuristic Thechniques

Taking into account the classifications introduced in the previous subsection, this subsection briefly introduces the most well-known metaheuristic techniques proposed in the literature.

The class of trajectory-based metaheuristics

includes several well-known techniques dating from the early years of the research on metaheuristics:

• **Simulated Annealing SA**: A trajectory-based search technique inspired on the annealing process of metals, which probabilistically allows accepting (i.e., moving to) a solution that

does not improve the objective function value, in order to escape from local optima (Kirkpatrick et al., 1983; Černý, 1985). SA is related to the Metropolis-Hasting Monte Carlo method, and it was the first metaheuristic proposed in the early 1980s, although the term 'metaheuristic' was not known in those years. SA is trajectory-based, applies a local-search, and is a memory-less metaheuristic.

• **Tabu Search TS**: A metaheuristic that applies a traditional local search strategy, enhanced by using memory structures (i.e., the 'tabu list') to store information about solutions visited in the past. Returning to recently visited solutions is not allowed, in order to promote diversification. TS was introduced by Glover (1986), in the article that first include the term 'metaheuristic', and later formalised by Glover (1989; 1990). TS is trajectory-based, applies a local-search, and is a memory-based metaheuristic.

• **Greedy randomised adaptive search procedure (GRASP)**: A constructivemeta-heuristic that performs a solution construction phase by selecting components following a greedy strategy, and then applies a local search in order to improve the constructed solution. GRASP was originally proposed by Feo and Resende (1989) and later formalised by Feo and Resende (1995). GRASP is trajectory-based, applies a constructive search, and is a memory-less meta-heuristic.

• **Variable neighbourhood search (VNS)**: a set of local search techniques based on the concept of using different neighbourhood structures during the search (Mladenovic and Hansen, 1997). A number of specific metaheuristics are within this class, including variable neighbourhood descent, basic VNS, and skewed VNS (Hansen and Mladenovic, 2002). VNS is trajectory-based, applies a local-search, and is a memory-less metaheuristic.

• **Iterated local search (ILS)**: A method that combines a hill-climbing/local-search tofind local optima and a stochastic perturbation or restart strategy, applying a simple and intuitive idea to prevent the search getting stuck in local optima (Lourenço et al.,2002). ILS is trajectory-based, applies a local-search, and is a memory-less metaheuristic.[14]

The class of population-based metaheuristic

includes many search techniques that use a group of search agents to benefit of cooperation and the (intrinsically) parallelism in the search procedure. Most population-based metaheuristic are

inspired on, or are based on emulating natural processes:[14]

• **Evolutionary computation (EC)**: A set of non-deterministic methods for search and optimization that emulate the evolutionary process of species in nature. EC include several approaches, among them the well-known evolutionary algorithms (EA), genetic programming (GP), and evolution strategies (ES). The main ideas for applying evolution and self-replication in computer science dates from the early 1960s (Rechenberg, 1965; Fogel et al., 1966); later EAs were first formulated by Holland (1975) and gently presented by Goldberg (1989). Since 1990, a large community of researchers has been studying and applying EC techniques. In general, EC methods are iterative search techniques (each iteration is called a generation) that work by applying stochastic operators on a set of candidate solutions (the population), in order to improve the fitness of solutions, a metric that evaluates how good a solution is to solve the problem. Different variation operators, most notably including the recombination of solutions and random changes (mutations) in their contents are applied for building new solutions during the search. The search is guided by a selection-of-the-best technique to tentative solutions of higher quality along the generations. In the last 25 years, EC techniques have been extensively applied to solve a large number of problems in many application areas.[14]

• **Swarm intelligence(SI)**: A class of metaheuristics based on the main concepts of cooperation from the collective behaviour of self-organised systems. They are characterised by using a set of entities or agents, which perform local search explorations in a neighbourhood, while they locally interact with both neighbouring agents and the environment. Although agents usually have limited search capabilities, an 'intelligent' search pattern emerges when considering large sets of agents, or swarms. The most well-known metaheuristics in this class are ACO, which emulates the behaviour of ants foraging for food (Dorigo, 1992), PSO, simulating the behaviour of flock of birds (Kennedy and Eberhart, 1995), and artificial immune system, which emulates its biological counterpart (DasGupta, 1998). Recently, some other metaheuristics based on the paradigm of swarm intelligence have been proposed, including bacterial foraging optimization (Passino, 2002), fish swarm optimization (Li et al., 2002), glowworm swarm optimization (Krishnanand and Ghose, 2005), firefly algorithm (Yang, 2008), Cuckoo search (Yang and Deb, 2009), and several bee algorithms that emulate the food foraging behaviour of bee swarms: honey bee algorithm (Nakrani and Tovey, 2004), honey-bee mating optimization (Haddad et al., 2006), and artificial bee colony (Karaboga and Basturk, 2007). Some of these new proposals of swarm metaheuristic algorithms are not free from criticism from the

community .[14]

• **Evolutionary inspired metaheuristics (EI)**: this class includes several other methods that apply a kind of population-based evolutionary search, with different features and search procedures. The most well-known metaheuristics in this class are: estimation of distribution algorithm (EDA), which applies a learning process based on building and sampling probabilistic models of promising candidate solutions (Larrañaga and Lozano, 2002); scatter search/path relinking (SS/PR), which applies an approach similar to evolutionary computation, but including a specific recombination and some ideas from TS (Glover, 1999); differential evolution (DE), specialised for optimizing real-valued functions (Storn and Price, 1997); harmony search (HS), applying an analogy of the improvisation of jazz musicians (Geem et al., 2001); and memetic algorithms (MA), evolutionary-based optimization techniques that include a local search phase after applying the evolutionary operators (Moscato, 1999). MA is often characterised as an hyper-heuristic algorithm, since they include a learning process to automate the search process.[14]

this project we will be focusing our experiments in population based metaheuristics, therefore in the next section we will provide more indepth details about this class ,which consists of nature/bio inspired techniques for problem solving and optimization.

2.3 Nature-inspired Optimization

Nature-inspired algorithms for optimization form a hot topic in the developments of new algorithms inspired by nature. These nature-inspired metaheuristic algorithms can be based on swarm intelligence, biological systems, physical and chemical systems. Therefore, these algorithms can be called swarm-intelligence-based,bio-inspired, physics-based and chemistry-based, depending on the sources of inspiration. Though not all of them are efficient, a few algorithms have proved to be very efficient and thus have become popular tools for solving real-world problems. Although some algorithms are insufficiently studied, **in this section we will present a comprehensive list of well known algorithms in the litterature as well as an indepth disscussion about new techniques which will be used in this project**

2.3.1 Source of inspiration

Nature has inspired many researchers in many ways and thus is a rich source of inspiration. Nowadays, most new algorithms are nature-inspired, because they have been developed by drawing inspiration from nature. Even with the emphasis on the source of inspiration, we can still have different levels of classifications, depending on how details and how many subsources we will wish to use. For simplicity, we will use the highest level sources such as biology, physics or chemistry. In the most generic term, the main source of inspiration is Nature. Therefore, almost all new algorithms can be referred to as nature-inspired. By far the majority of nature-inspired algorithms are based on some successful characteristics of biological system. Therefore, the largest fraction of nature-inspired algorithms are biology-inspired, or bio-inspired for short. Among bio-inspired algorithms, a special class of algorithms have been developed by drawing inspiration from swarm intelligence. Therefore, some of the bioinspired algorithms can be called swarm-intelligence based. In fact, algorithms based on swarm intelligence are among the most popular. Good examples are ant colony optimization , particle swarm optimization , cuckoo search , bat algorithm , and firefly algorithm .

Obviously, not all algorithms were based on biological systems. Many algorithms have been developed by using inspiration from physical and chemical systems. Some may even be based on music.[9]

2.3.2 Classifications of Algorithms

Based on the above discussions, we can divide all existing algorithms into four major categories:

- swarm intelligence (SI) based.
- bio-inspired (but not SI-based).
- physics/chemistry-based.
- and others.

We will summarize them in the rest of this section. However, we will focus here on the relatively new algorithms.

Swarm intelligence based

Swarm intelligence (SI) concerns the collective, emerging behaviour of multiple, interacting agents who follow some simple rules. While each agent may be considered as unintelligent, the whole system of multiple agents may show some self-organization behaviour and thus can behave like some sort of collective intelligence. Many algorithms have been developed by drawing inspiration from swarm-intelligence systems in nature. All SI-based algorithms use multi-agents, inspired by the collective behaviour of social insects, like ants, termites, bees, and wasps, as well as from other animal societies like flocks of birds or fish.

- **Particle swarm optimization (PSO)** : uses the swarming behaviour of fish and birds.
- **Firefly algorithm (FA)** : uses the flashing behaviour of swarming fireflies.
- **Cuckoo search (CS)** is based on the brooding parasitism of some cuckoo species.
- **bat algorithm(BA)** : uses the echolocation of foraging bats.
- **Ant colony optimization(ACO)** uses the interaction of social insects (e.g., ants).
- **Bee colony optimization** : are all based on the foraging behaviour of honey bees.

SI-based algorithms are among the most popular and widely used. There are many reasons for such popularity, one of the reasons is that SI-based algorithms usually sharing information among multiple agents, so that selforganization, co-evolution and learning during iterations may help to provide the high efficiency of most SI-based algorithms. Another reason is that multiple agent can be parallelized easily so that large-scale optimization becomes more practical from the implementation point of view.[9]

Bio-Inspired (but not SI-Inspired)

SI-based algorithms belong to a wider class of algorithms, called bio-inspired algorithms. In fact, bio-inspired algorithms form a majority of all nature-inspired algorithms. From the set theory point of view, SI-based algorithms are a subset of bio-inspired algorithms, while bio-inspired algorithms are a subset of nature-inspired algorithms. That is SI-based $\subset$ bio-inspired $\subset$nature-inspired.

Conversely, not all nature-inspired algorithms are bioinspired, and some are purely physics and chemistry based algorithms as we will see below. Many bio-inspired algorithms do not use directly the swarming behaviour. Therefore, it is better to call them bio-inspired, but not SI-based. For example, genetic algorithms are bio-inspired, but not SI-based. However, it is not easy to classify certain algorithms such as differential evolution (DE). Strictly speaking, DE is not bio-inspired because there is no direct link to any biological behaviour. However, as it has some similarity to genetic algorithms and also has a key word 'evolution', we tentatively put it in the category of bioinspired algorithms. For example, the flower algorithm, or flower pollination algorithm , developed by Xin-She Yang in 2012 is a bio-inspired algorithm, but it is not a SI-based algorithm.

• **flower pollination algorithm** : tries to mimic the pollination characteristics of flowering plants and the associated flower consistency of some pollinating insects.[9]

Physics and Chemistry Based

Not all metaheuristic algorithms are bio-inspired, because their sources of inspiration often come from physics and chemistry. For the algorithms that are not bio-inspired, most have been developed by mimicking certain physical and/or chemical laws, including electrical charges, gravity, river systems, etc. As different natural systems are relevant to this category, we can even subdivide these into many subcategories which is not necessary. Schematically, we can represent the relationship of physics and chemistry based algorithms as the follows:

$$\frac{\text{Physics algorithms}}{\text{Chemistry algorithms}} \begin{cases} \notin \text{bio-inspired algorithms} \\ \in \text{nature-inspired algorithms} \end{cases}$$

Figure 2.2: The relationship of physics and chemistry based algorithms[9]

Though physics and chemistry are two different subjects, however, it is not useful to subdivide this subcategory further into physics-based and chemistry. After all, many fundamental laws are the same. So we simply group them as physics and chemistry based algorithms.[9]

Other Algorithms

When researchers develop new algorithms, some may look for inspiration away from nature. Consequently, some algorithms are not bio-inspired or physics/chemistry-based, it is sometimes difficult to put some algorithms in the above three categories, because these algorithms have been developed by using various characteristics from different sources, such as social, emotional, etc. In this case, it is better to put them in the other category.[9]

In the next section we will discuss the details of the optimization techniques used in our project which are:

- **Dragonfly Algorithm (DA 2015)**.
- **Grey Wolf Optimization (GWO 2014)**.

2.4 Dragonfly Algorithm DA

Dragonfly algorithm (DA) is a novel swarm intelligence optimization technique . Proposed by Mirjalili Sayedali. The main inspiration of the DA algorithm originates from the static and dynamic swarming behaviours of dragonflies in nature. Two essential phases of optimization, exploration and exploitation, are designed by modelling the social interaction of dragonflies in navigating, searching for foods, and avoiding enemies when swarming dynamically or statistically.[18]

2.4.1 Operators for exploration and exploitation

According to Reynolds, the behaviour of swarms follows three primitive principles :

- **Separation**, which refers to the static collision avoidance of the individuals from other individuals in the neighbourhood.
- **Alignment**, which indicates velocity matching of individuals to that of other individuals in neighbourhood.
- **Cohesion**, which refers to the tendency of individuals towards the centre of the mass of

the neighbourhood.

The main objective of any swarm is survival, so all of the individuals should be attracted towards food sources and distracted outward enemies. Considering these two behaviours, there are five main factors in position updating of individuals in swarms as shown in Each of these behaviours is mathematically modelled as follows:

• The separation is calculated as follows :

$$S_i = -\sum_{j=1}^{N} X - X_j$$

(Eq.1)

where X is the position of the current individual, Xj shows the position j-th neighbouring individual, and N is the number of neighbouring individuals.

• The Alignment is calculated as follows :

$$A_i = \frac{\sum_{j=1}^{N} V_j}{N}$$

(Eq.2)

where Xj shows the velocity of j-th neighbouring individual.

• The Cohesion is calculated as follows:

$$C_i = \frac{\sum_{j=1}^{N} X_j}{N} - X$$

(Eq.3)

where X is the position of the current individual, N is the number of neighbourhoods, and Xj shows the position j-th neighbouring individual.

• Attraction towards a food source is calculated as follows:

$$F_i = X^{+} - X$$

(Eq.4)

where X is the position of the current individual, and X+ shows the position of the food

source.

- Distraction outwards an enemy is calculated as follows:

$$E_i = X^- + X$$

(Eq.5)

where X is the position of the current individual, and X- shows the position of the enemy.

The behaviour of dragonflies is assumed to be the combination of these five corrective patterns . as shown in

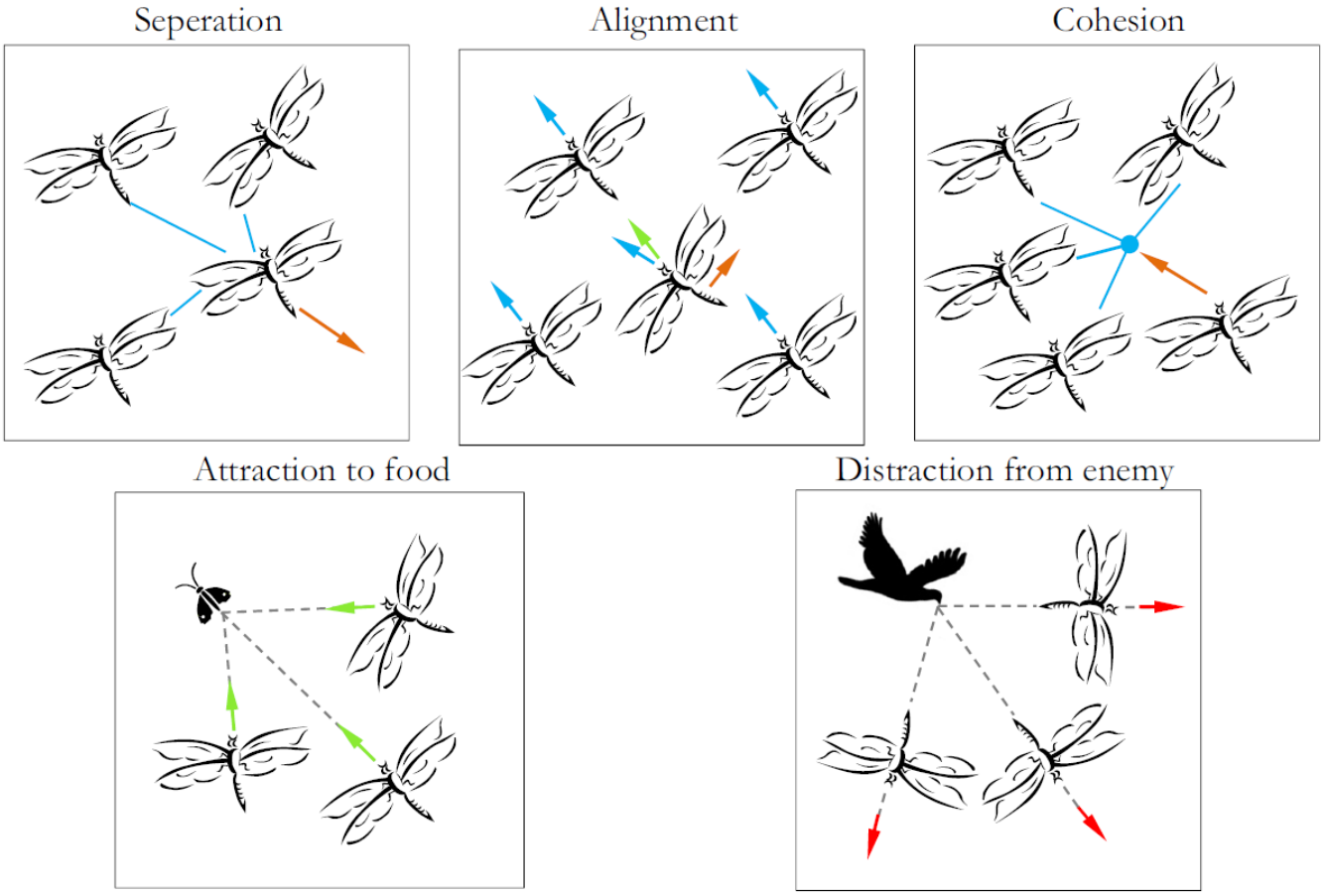

Figure 2.3: Primitive corrective patterns between individuals in a swarm[18]

To update the position of artificial dragonflies in a search space and simulate their movements, two vectors are considered: step (Δ X) and position (X). The step vector is analogous to the velocity vector in PSO, and the DA algorithm is developed based on the framework of the PSO algorithm. The step vector shows the direction of the movement of the dragonflies

and defined as follows (note that the position updating model of artificial dragonflies is defined in one dimension, but the introduced method can be extended to higher dimensions):

$$\Delta X_{t+1} = (sS_i + aA_i + cC_i + fF_i + eE_i) + w\Delta X_t$$

(Eq.6)

where s shows the separation weight, Si indicates theseparation of the i-th individual, a is the alignment weight, A is the alignment of i-th individual, c indicates the cohesion weight, Ci is the cohesion of the i-th individual, f is the food factor, Fi is the food source of the i-th individual, e is the enemy factor, Ei is the position of enemy of the i-th individual, w is the inertia weight, and t is the iteration counter. After calculating the step vector, the position vectors are calculated as follows:

$$X_{t+1} = X_t + \Delta X_{t+1}$$

(Eq.7)

where t is the current iteration. With separation, alignment, cohesion, food, and enemy factors (s, a, c, f, and e), different explorative and exploitative behaviours can achieved during optimization. Neighbours of dragonflies are very important, so a neighbourhood (circle in a 2D, sphere in a 3D space, or hypersphere in an nD space) with a certain radius is assumed around each artificial dragonfly. An example of swarming behaviour of dragonflies with increasing neighbourhood radius using the proposed mathematical model is illustrated in the Figure.

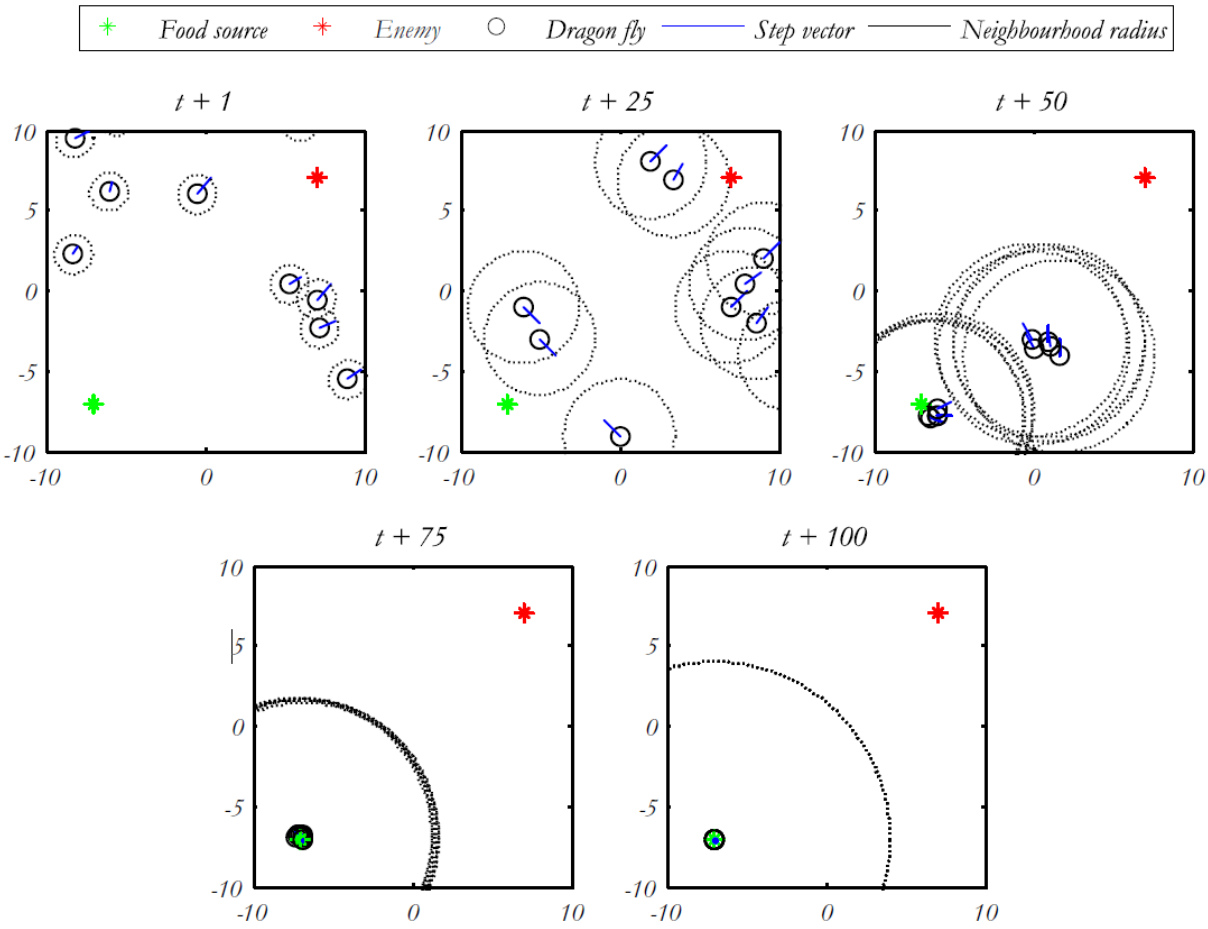

Figure 2.4: Swarming behaviour of Enemy artificial dragon flies (w = 0.9–0.2, s = 0.1, a = 0.1, c = 0.7, f = 1, e = 1)[18]

As discussed in the previous subsection, dragonflies only show two types of swarms: static and dynamic as shown in Figure. 14. As may be seen in this figure, dragonflies tend to align their flying while maintaining proper separation and cohesion in a dynamic swarm. In a static swarm, however, alignments are very low while cohesion is high to attack preys. Therefore, we assign dragonflies with high alignment and low cohesion weights when exploring the search space and low alignment and high cohesion when exploiting the search space. For transition between exploration and exploitation, the radii of neighbourhoods are increased proportional to the number of iterations. Another way to balance exploration and exploitation is to adaptively tune the swarming factors (s, a, c, f, e, and w) during optimization.

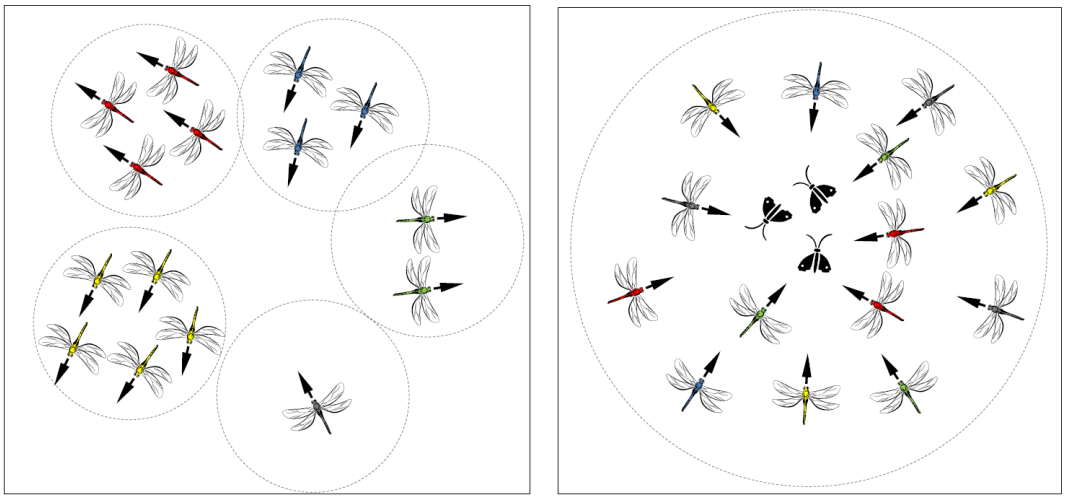

Figure 2.5: Dynamic versus static dragonfly swarms[18]

A question may rise here as to how the convergence of dragonflies is guaranteed during optimization. The dragonflies are required to change their weights adaptively for transiting from exploration to exploitation of the search space. It is also assumed that dragonflies tend to see more dragonflies to adjust flying path as optimization process progresses. In other word, the neighbourhood area is increased as well whereby the swarm become one group at the final stage of optimization to converge to the global optimum. The food source and enemy are chosen from the best and worst solutions that the whole swarm is found so far. This causes convergence towards promising areas of the search space and divergence outward non-promising regions of the search space.

To improve the randomness, stochastic behaviour, and exploration of the artificial dragonflies, they are required to fly around the search space using a random walk (Levy flight) when there is no neighbouring solutions. In this case, the position of dragonflies is updated using the following equation:

$$X_{t+1} = X_t + \text{Lévy}(d) \times X_t$$

(Eq.8)

where t is the current iteration, and d is the dimension of the position vectors.

The Levy flight is calculated as follows :

$$\text{Lévy}(x) = 0.01 \times \frac{r_1 \times \sigma}{|r_2|^{\frac{1}{\beta}}}$$

(Eq.9)

where r1, r2 are two random numbers in [0,1], β is a constant (equal to 1.5 in this work), and α is calculated as follows:

$$\sigma = \left(\frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\left(\frac{\beta-1}{2}\right)}} \right)^{1/\beta}$$

(Eq.10)

where $\Gamma(x) = (x-1)!$.

The DA algorithm starts optimization process by creating a set of random solutions for a given optimization problems. In fact, the position and step vectors of dragonflies are initialized by random values defined within the lower and upper bounds of the variables. In each iteration, the position and step of each dragonfly are updated using Eqs. (7)/(8) and (6). For updating X and DX vectors, neighbourhood of each dragonfly is chosen by calculating the Euclidean distance between all the dragonflies and selecting N of them. The position updating process is continued iteratively until the end criterion is satisfied.[18] The pseudo-codes of the DA algorithm are provided in the next Figure.

```
Initialize the dragonflies population Xi (i = 1, 2, ..., n)
Initialize step vectors ΔXi (i = 1, 2, ..., n)
while the end condition is not satisfied
        Calculate the objective values of all dragonflies
        Update the food source and enemy
        Update w, s, a, c, f, and e
        Calculate S, A, C, F, and E using Eqs. (3.1) to (3.5)
        Update neighbouring radius
        if a dragonfly has at least one  neighbouring dragonfly
                Update velocity vector using Eq. (3.6)
                Update position vector using Eq. (3.7)
        else
                Update position vector using Eq. (3.8)
        end if
        Check and correct the new positions based on the
        boundaries of variables
end while
```

Figure 2.6: Dragonfly algorithm pseudo-code[18]

2.5 Grey Wolf Optimization GWO

The GWO algorithm mimics the leadership hierarchy and hunting mechanism of gray wolves in nature proposed by Mirjalili et al. in 2014. Four types of grey wolves such as alpha, beta, delta, and omega are employed for simulating the leadership hierarchy. In addition, three main steps of hunting, searching for prey, encircling prey, and attacking prey, are implemented to perform optimization.[19]

2.5.1 Mathematical Model

The hunting technique and the social hierarchy of grey wolves are mathematically modeled in order to design GWO and perform optimization. The proposed mathematical models of the social hierarchy, tracking, encircling, and attacking prey are as follows:

Social hierarchy

In order to mathematically model the social hierarchy of wolves when designing GWO, we consider the fittest solution as the alpha (α). Consequently, the second and third best solutions are named beta (β) and delta (δ) respectively. The rest of the candidate solutions are assumed to be omega (ω). In the GWO algorithm the hunting (optimization) is guided by α, β, and δ. The ω wolves follow these three wolves.

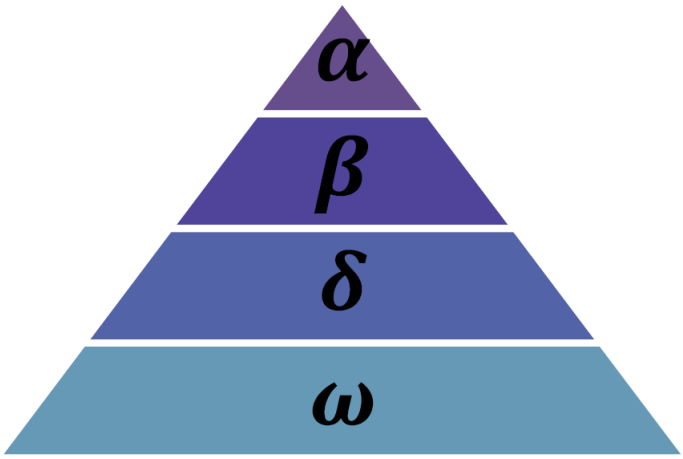

Figure 2.7: Hierarchy of grey wolf (dominanc decreases from top down).[19]

Encyvling Prey

As mentioned above, grey wolves encircle prey during the hunt. In order to mathematically model encircling behavior the following equations are proposed:

$$\vec{D} = |\vec{C}.\overrightarrow{X_p}(t) - \vec{X}(t)|$$
$$\vec{X}(t+1) = \overrightarrow{X_p}(t) - \vec{A}.\vec{D}$$

where t indicates the current iteration,$\overrightarrow{A}$ and $\overrightarrow{C}$ are coefficient vectors, $\overrightarrow{X}$p the position vector of the prey, and $\overrightarrow{X}$ indicates the position vector of a grey wolf. The vector $\overrightarrow{A}$ and $\overrightarrow{C}$ are calculated as follows:

$$\vec{A} = 2\vec{a}.\overrightarrow{r_1} - \vec{a}$$
$$\vec{C} = 2.\overrightarrow{r_2}$$

Where components of $\overrightarrow{\alpha}$ are linearly decreased from 2 to 0 over the course of iterations and r1 ,r2 are random vectors in [0,1]. With the above equations, a grey wolf in the position of (X,Y) can update its position according to the position of the prey (X*,Y*). Different places around the best agent can be reached with respect to the current position by adjusting the value of r1 and r2 vectors. For instance, (X*-X,Y*) can be reached by setting $\overrightarrow{\alpha}$=(1,0) and $\overrightarrow{C}$=(1,1) . Note that the random vectors $\overrightarrow{A}$ and $\overrightarrow{C}$ allow wolves to reach any position between the two particular points. So a grey wolf can update its position inside the space around the prey in any random location by the above-mentioned equations. The same concept can be extended to a search space with n dimensions, and the grey wolves will move in hyper-cubes (or hyper-spheres) around the best solution obtained so far.

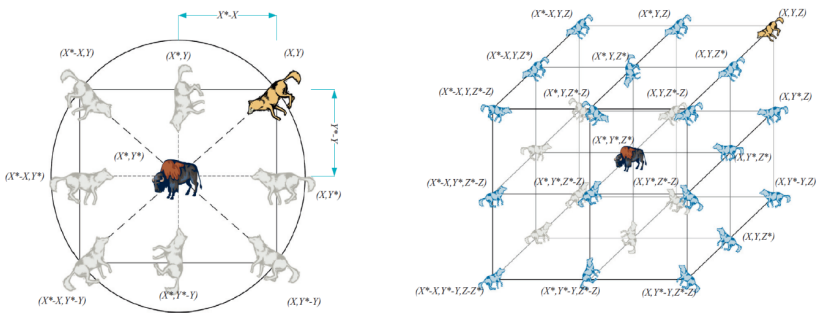

Figure 2.8: 2D and 3D position vectors and their possible next locations.[19]

Hunting

Grey wolves have the ability to recognize the location of prey and encircle them. The hunt is usually guided by the alpha. The beta and delta might also participate in hunting occasionally. However, in an abstract search space we have no idea about the location of the optimum (prey). In order to mathematically simulate the hunting behavior of grey wolves, we suppose that the alpha (best candidate solution) beta, and delta have better knowledge about the potential location of prey. Therefore, we save the first three best solutions obtained so far and oblige the other search agents (including the omegas) to update their positions according to the position of the best search agent. The following formulas are proposed in this regard.

$$\vec{D_\alpha} = |\vec{C_1}.\vec{X_\alpha} - \vec{X}|$$
$$\vec{D_\beta} = |\vec{C_2}.\vec{X_\beta} - \vec{X}|$$
$$\vec{D_\delta} = |\vec{C_3}.\vec{X_\delta} - \vec{X}|$$
$$\vec{X_1} = \vec{X_\alpha} - \vec{A_1}.(\vec{D_\alpha})$$
$$\vec{X_2} = \vec{X_\beta} - \vec{A_2}.(\vec{D_\beta})$$
$$\vec{X_3} = \vec{X_\delta} - \vec{A_3}.(\vec{D_\delta})$$
$$\vec{X}(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3}$$

With these equations, a search agent updates its position according to alpha, beta, and delta in a n dimensional search space. In addition, the final position would be in a random place within a circle which is defined by the positions of alpha, beta, and delta in the search space. In other words alpha, beta, and delta estimate the position of the prey, and other wolves updates their positions randomly around the prey.

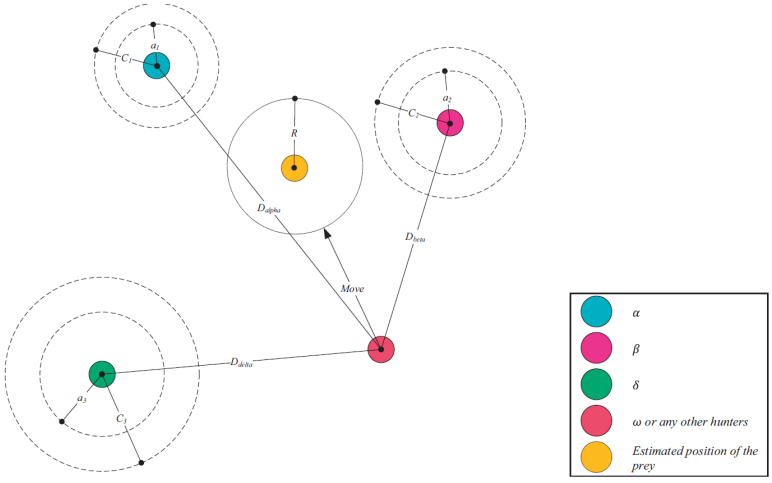

Figure 2.9: Position updating in GWO.[19]

Attacking Prey(Exploitation)

As mentioned above the grey wolves finish the hunt by attacking the prey when it stops moving. In order to mathematically model approaching the prey we decrease the value of $\vec{\alpha}$. Note that the fluctuation range of $\vec{A}$ is also decreased by $\vec{\alpha}$. In other words $\vec{A}$ is a random value in the interval [-2a,2a] where a is decreased from 2 to 0 over the course of iterations. When random values of $\vec{\alpha}$ are in [-1,1], the next position of a search agent can be in any position between its current position and the position of the prey. With the operators proposed so far, the GWO algorithm allows its search agents to update their position based on the location of the alpha, beta, and delta; and attack towards the prey. However, the GWO algorithm is prone to stagnation in local solutions with these operators. It is true that the encircling mechanism proposed shows exploration to some extent, but GWO needs more operators to emphasize exploration.

Figure 2.10: Hunting behavior of grey wolves: (A) chasing, approaching, and tracking prey (B–D) pursuiting, harassing, and encircling (E) stationary situation and attack.[19]

Searching for Prey

Grey wolves mostly search according to the position of the alpha, beta, and delta. They diverge from each other to search for prey and converge to attack prey. In order to mathematically

model divergence, we utilize $\vec{A}$ with random values greater than 1 or less than -1 to oblige the search agent to diverge from the prey. This emphasizes exploration and allows the GWO algorithm to search globally. | A | >1 forces the grey wolves to diverge from the prey to hopefully find a fitter prey. Another component of GWO that favors exploration is $\vec{C}$, which contains random values in [0, 2]. This component provides random weights for prey in order to stochastically emphasize (C>1) or de-emphasize (C<1) the effect of prey in defining the distance. This assists GWO to show a more random behavior throughout optimization, favoring exploration and local optima avoidance. It is worth mentioning here that C is not linearly decreased in contrast to A. We deliberately require C to provide random values at all times in order to emphasize exploration not only during initial iterations but also final iterations. This component is very helpful in case of local optima stagnation, especially in the final iterations.

The C vector can be also considered as the effect of obstacles to approaching prey in nature. Generally speaking, the obstacles in nature appear in the hunting paths of wolves and in fact prevent them from quickly and conveniently approaching prey. This is exactly what the vector C does. Depending on the position of a wolf, it can randomly give the prey a weight and make it harder and farther to reach for wolves, or vice versa. To sum up, the search process starts with creating a random population of grey wolves (candidate solutions) in the GWO algorithm. Over the course of iterations, alpha, beta, and delta wolves estimate the probable position of the prey. Each candidate solution updates its distance from the prey. The parameter a is decreased from 2 to 0 in order to emphasize exploration and exploitation, respectively. Candidate solutions tend to diverge from the prey when | $\vec{A}$ | >1 and converge towards the prey when | $\vec{A}$ | <1. Finally, the GWO algorithm is terminated by the satisfaction of an end criterion.[19]

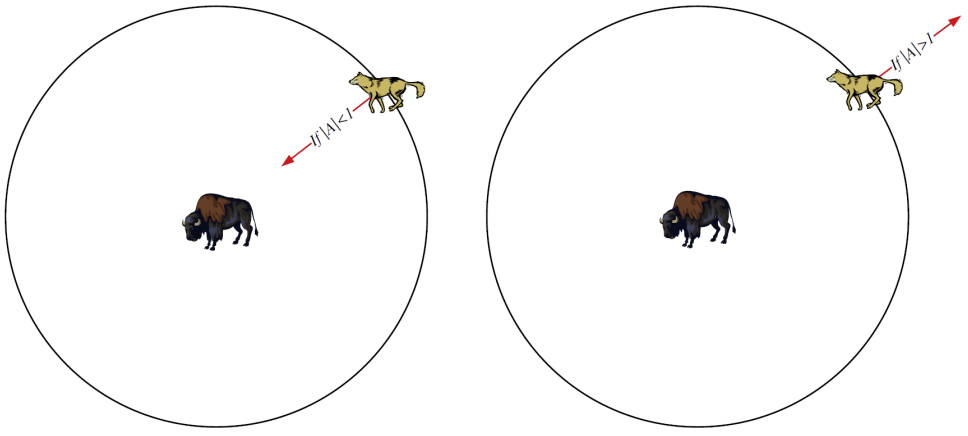

Figure 2.11: Atacking prey versus searching for prey[19]

2.5.2 GWO pseudo-code

- Initialize the grey wolf population Xi (i = 1, 2, ..., n)
- Initialize a, A, and C
- Calculate the fitness of each search agent
- X_α =the best search agent
- X_β =the second best search agent
- X_δ =the third best search agent
  - while (t < Max number of iterations)
    - for each search agent
      - Update the position of the current search agent by above equations
    - end for
    - Update a, A, and C
    - Calculate the fitness of all search agents
    - Update X_α, X_β, and X_δ
    - t=t+1
  - end while
- return X_α

Figure 2.12: GWO pseudo-code [19]

To see how GWO is theoretically able to solve optimization problems, some points may be noted:

• The proposed social hierarchy assists GWO to save the best solutions obtained so far over the course of iteration

• The proposed encircling mechanism defines a circle-shaped neighborhood around the solutions which can be extended to higher dimensions as a hyper-sphere

• The random parameters A and C assist candidate solutions to have hyper-spheres with different random radii

• The proposed hunting method allows candidate solutions to locate the probable position of the prey

• Exploration and exploitation are guaranteed by the adaptive values of a and A

• The adaptive values of parameters a and A allow GWO to smoothly transition between exploration and exploitation

• With decreasing A, half of the iterations are devoted to exploration ($| A | \geq 1$) and the other half are dedicated to exploitation ($| A | < 1$)

• The GWO has only two main parameters to be adjusted (a and C)[19]

2.6 Summary

In this Chapter we had an introduction about Metaheurstics and Optimization Algorithms as well as Nature-Inspired Optimization Algorithms. And then we took a deep dive into Dragonfly algorithm. DA is inspired by the behaviour of dragonflies' swarms in nature. Static and dynamic swarming behaviours of dragonflies were used to explore and exploit the search space, respectively.

The DA algorithm was equipped with five parameters to control cohesion, alignment, separation, attraction (towards food sources), and distraction (outwards enemies) of individuals in the swarm. Suitable operators were integrated to the proposed DA algorithm for solving different kind of problems.

We Also talked about another novel nature-inspired Optimization Algorithms which is Grey wolf Optimization (GWO) , GWO mimics the leadership hierarchy and hunting mechanism of grey wolves in nature. Four types of grey wolves such as alpha, beta, delta, and omega are employed for simulating the leadership hierarchy. In addition to, the three main steps of hunting, searching for prey, encircling prey, and attacking prey.

The following two chapters will be the design and implementation of this project regarding training neural networks with novel nature inspired techniuqes for medical prediction.

Chapter 3

Project Design

3.1 Introduction

Medical prediction has attracted a lot of attention in recent decades. Researchers have made significant breakthroughs in this area with the rapid development of maching learning algorithms and the most successful applications in this field. So far, they are using neural network-based methods. Artificial neural networks or ANN have been very successful in various medical prediction applications.

In our project, we were interested in creating a medical prediction system that uses ANNs. To achieve our goal, In this chapter we propose a novel technique for training and optimizing neural networks for better performance and prediction accuracy using the new nature based optimization algorithm (Dragonfly algorithms), this method is called (ANN-DA) .

NOTE :In this project we will be training and optimizing feed-forward neural networks from a simple multilayer perceptron MLP to deep convolutional neural networks (ResNet-50 CNN).that is why we chose the term ANN-DA because it generalize Artificial neural networks optimized by dragonfly algorithm

3.2 System Design

3.2.1 Methodology

In our system, we are going to use a few of the image processing techniques such as pre-processing, adjustments, and data augmentation etc.., also we gonna use an appropriate supervised machine learning methods both deep and shallow (simple MLP to deep CNN) which are very suitable for medical prediction and image recognition. And for training and optimizing our ANN we will be using the novel nature-based Agorithm called Dragonfly Algorithm (DA).

This technique allows us to make medical predcition which we want to accomplish by creating and designing a descent training architecture seeking for better accuracy results.

3.2.2 Global system design

Generally, Our medical prediction system will be following a certain steps, as we represent the global architecture in the below figure:

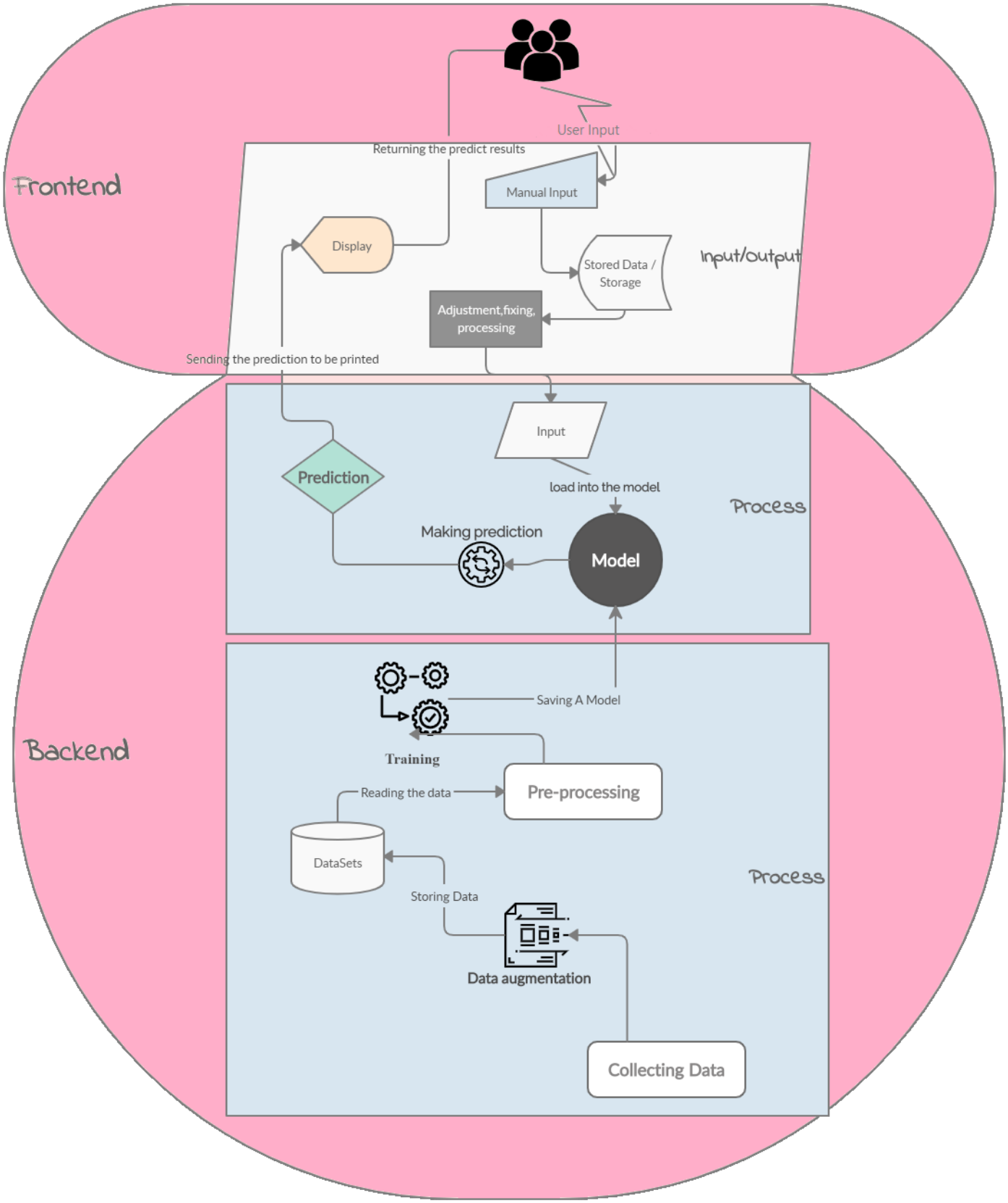

Figure 3.1: Global project design

As shown in the previous figure, our system have two main phases :

Front-end : (User Inteface) which is an important phase for the user where the interaction and the requests of files are done, also the displaying and the stored data and results after getting them from the back-end phase.

Back-end : is the most important phase in the system, during this process we will do the data collecting including data pre-processing and augmentation, and then the training phase which gives us as an output the model that we gonna use later-on the prediction phase, the prediction model takes as input data and then results as output.

3.2.3 Detailed system design

Back-end

The back-end phase its the main body of the system where we gonna pass thought important steps described in following :

1. The collection and Preparation of datasets.
2. Pre-processing the datasets.
3. Training.
4. Use of the output model.

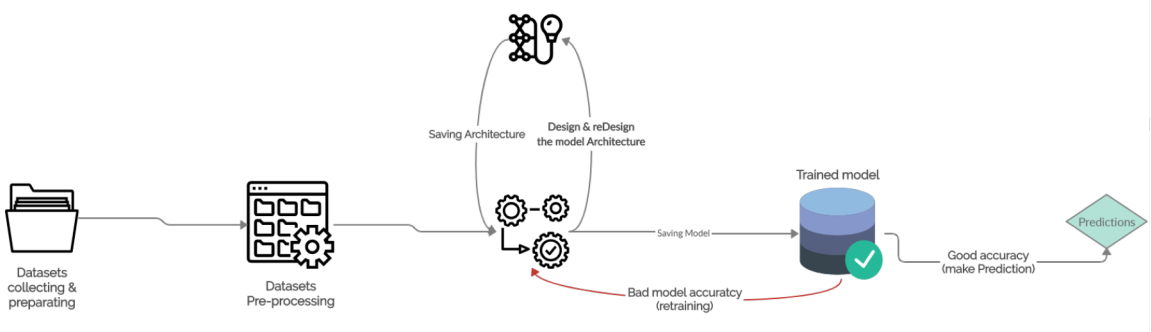

Figure 3.2: Back-end phase design[3]

A-The collection and preparation of datasets In our system we are going to work on four different type of diseases (breast cancer, heart disease, hepatitis, and Covid-19), for that we need to collect and prepare the breast cancer, heart disease and hepatitis datasets for our MLP, wich are textual datasets, and the COVID-19 dataset for our CNN which is a set of Lungs X-Ray images

Breast cancer : This dataset was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg.[24]

- Number of Instances: 699 (Training set : 599, Testing set: 100)
- Number of Attributes: 10 plus the class attribute :

1. Sample code number	id number
2. Clump Thickness	1 - 10
3. Uniformity of Cell Size	1 - 10
4. Uniformity of Cell Shape	1 - 10
5. Marginal Adhesion	1 - 10
6. Single Epithelial Cell Size	1 - 10
7. Bare Nuclei	1 - 10
8. Bland Chromatin	1 - 10
9. Normal Nucleoli	1 - 10
10. Mitoses	1 - 10
11. Class:	(0 for benign, 1 for malignant)

Table 3.1: Breast cancer dataset attributes table[24]

- Class distribution: Benign: 458 (65.5%) Malignant: 241 (34.5%)

Heart disease : The dataset describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT) images. Each of the patients is classified into two categories: normal and abnormal. The dataset of 267 SPECT image sets (patients) was processed to extract features that summarize the original SPECT images. As a result, 44 continuous feature pattern was created for each patient. The pattern was further processed to obtain 22 binary feature patterns.

SPECT is a good data set for testing ML algorithms; it has 267 instances that are descibed by 23 binary attributes

• Attribute information :

A1 : Overall diagnosis (1,0 binary, class attribute)

A2 : 0,1 (the partial diagnosis 1, binary)

Ai : 0,1 (the partial diagnosis i, binary)

A23 : 0,1 (the partial diagnosis 23, binary)

• Original owners : Krzysztof J. Cios, Lukasz A. Kurgan, University of Colorado at Denver, Denver, CO 80217, U.S.A.

• Dataset is devides to :Training set 80 instances, and Testing set 187 instances[10]

Hepatitis : this dataset was optained from Carnegie-Mellon University in Pittsburgh, Pennsylvania. Via DR. G.Gong[26]

• Number of Instances: 648 (Training set : 568 Testing set : 80)

• Number of Attributes: 20 (including the class attribute)

• Attribute information:

1. Class:	DIE, LIVE
2. AGE:	10, 20, 30, 40, 50, 60, 70, 80
3. SEX:	male, female
4. STEROID:	no, yes
5. ANTIVIRALS:	no, yes
6. FATIGUE:	no, yes
7. MALAISE:	no, yes
8. ANOREXIA:	no, yes
9. LIVER BIG:	no, yes
10. LIVER FIRM:	no, yes
11. SPLEEN PALPABLE:	no, yes
12. SPIDERS:	no, yes
13. ASCITES:	no, yes
14. VARICES:	no, yes
15. BILIRUBIN:	0.39, 0.80, 1.20, 2.00, 3.00, 4.00
16. ALK PHOSPHATE:	33, 80, 120, 160, 200, 250
17. SGOT:	13, 100, 200, 300, 400, 500
18. ALBUMIN:	2.1, 3.0, 3.8, 4.5, 5.0, 6.0
19. PROTIME:	10, 20, 30, 40, 50, 60, 70, 80, 90
20. HISTOLOGY:	no, yes

Table 3.2: Hepatitis Dataset attributes table[26]

COVID-19 dataset utilized in this research was curated by Dr. Joseph Cohen, a post-doctoral fellow at the University of Montreal. Thanks to the article by Dr. Adrian Rosebrock for making this chest radiograph dataset reachable to researchers across the globe and for presenting the initial work using Deep Learning. we solely utilize the x-ray images. We should be able to download the images from the article directly. After downloading the ZIP files from the website and extracting them to a folder called "Covid 19", we have one sub-folder per class in "dataset". Label "Covid" indicates the presence of COVID-19 in the patient and "normal" otherwise. Since, we have equal distribution (25 images) of both classes, there is no class imbalance issue here.[16]

B-Pre-processing the datasets Data preprocessing is the first (and arguably most important) step toward building a working Maching learning model. It's critical . Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues.

In Real world data are generally incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data. Noisy: containing errors or outliers. Inconsistent: containing discrepancies in codes or names.

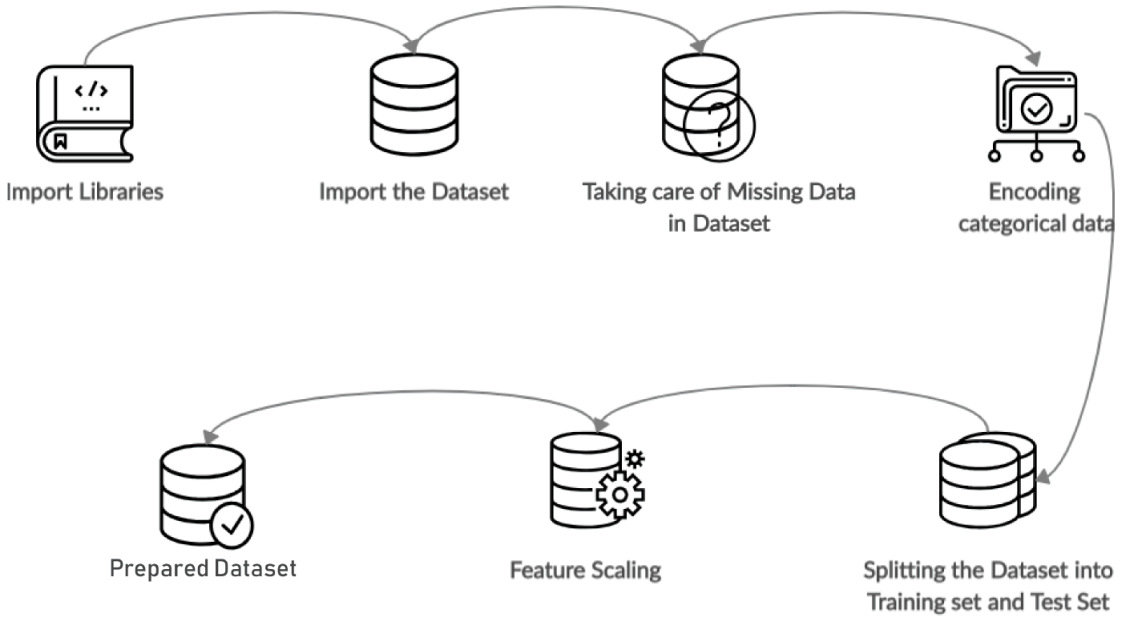

Figure 3.3: Pre-Processing steps[3]

Step 1: Import Libraries

First step is usually importing the libraries that will be needed in the program. A library is essentially a collection of modules that can be called and used. A lot of the things in the programming world do not need to be written explicitly ever time they are required. There are functions for them, which can simply be invoked. also the libraries we need to load datasets and work on them.

Step 2: Import the Datasets

A lot of datasets come in different formats. We will need to locate the directory of the

datasets files at first (it's more efficient to keep the dataset in the same directory your system) read them using a methods which can be found in the libraries into the variables on memory,furthermore used in following pre-processing steps.

Step 3: Taking care of Missing Data in Dataset

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously you could remove the entire line of data, of course we would not want to do that. One of the most common idea to handle the problem is to take a mean of all the values of the same column and have it to replace the missing data.

Step 4: Encoding categorical data

Sometimes our data is in qualitative form, that is we have texts as our data. We can find categories in text form. Now it gets complicated for machines to understand texts and process them, rather than numbers, since the models are based on mathematical equations and calculations. Therefore, we have to encode the categorical data.

Difficulty saving money	Counts	Frequencies
Very	231	46%
Somewhat	196	39%
Not very	58	12%
Not at all	14	3%
Not sure	1	0%
Total	500	100%

Figure 3.4: Example of categorial data[3]

ID	Gender
1	Male
2	Female
3	Not Specified
4	Not Specified
5	Female

—>

ID	Male	Female	Not Specified
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1
5	0	1	0

Figure 3.5: Example of dummy data[3]

Step 5: Splitting the Dataset into Training set and Test Set

Now we need to split our dataset into two sets–a Training set and a Test set. We will train our machine learning models on our training set, our machine learning models will try to understand any correlations in our training set and then we will test the models on our test set to check how accurately it can predict. A general rule of the thumb is to allocate 80% of the dataset to training set and the remaining 20% to test set.

Step 6: Feature Scaling

The final step of data preprocessing is to apply the very important feature scaling. Feature Scaling It is a method used to standardize the range of independent variables or features of data. also to limit the range of variables so that they can be compared on common grounds. Suppose we have this data-set:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	Bi
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Figure 3.6: Example of a dataset with undefined values[3]

In the figure above, notice the Age and Salary column. You can easily noticed Salary and Age variable don't have the same scale and this will cause some issue in your deep learning model. Because most of the Deep learning models are based on Euclidean Distance.

$$d(A, B) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

Let's say we take two values from Age and Salary column, Age- 40 and 27, Salary- 72000 and 48000.

One can easily compute and see that Salary column will be dominated in Euclidean Distance. And we don't want this thing. So there are several ways of scaling your data

Re-scaling (min-max normalization) Also known as min-max scaling or min-max normalization, is the simplest method and consists in re-scaling the range of features to scale

the range in [0, 1] or [-1, 1]. Selecting the target range depends on the nature of the data. The general formula is given as:

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

where x is an original value, x' is the normalized value. For example, suppose that we have the students' weight data, and the students' weights span [160 pounds, 200 pounds]. To re-scale this data, we first subtract 160 from each student's weight and divide the result by 40 (the difference between the maximum and minimum weights).[3]

Mean normalization

$$x' = \frac{x - average(x)}{max(x) - min(x)}$$

where x is an original value, x' is the normalized value.

Standardization The general method of calculation is to determine the distribution mean and standard deviation for each feature. Next we subtract the mean from each feature. Then we divide the values (mean is already subtracted) of each feature by its standard deviation.

$$x' = \frac{x - \bar{x}}{\sigma}$$

Where x is the original feature vector, x = average(x) x = average(x) is the mean of that feature vector, and sigma is its standard deviation.[3]

C- Training Its the important phase in our system, which we gonna train an Artificial neural networks model with a specific architecture wheeling to achieve best accuracy result and less data loss, Next can be used in Medical predition, the training phase could be divided into two steps illustrated in the figure

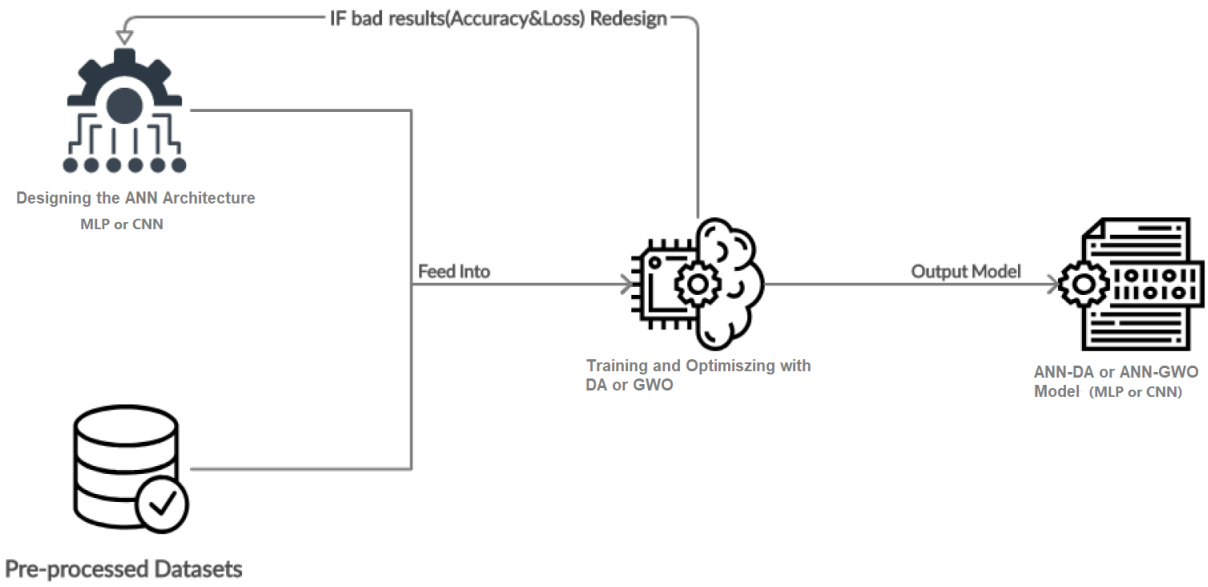

Figure 3.7: Training phase

Step 1: Model architecture design

As we already know the maching learning method we are going to work with "Artificial neural networks" (MLPs and deep CNNs) , We design the model parameters with how much layers, each layer size , each layer type and defining layer function , Also the layers placement.

Step 2: Training and Optimizing After getting the model architecture ready and designed, And the pre-processed datasets, Now we configure the training and optimizing phase : there are two cases

• **Case 1 : training and optimizing the ANN usnig the Dragonfly algorithm (ANN-DA) :**

1. Initialize the dragonfly positions X (weights solutions) to 0.5.

2. Initialize the positions difference (ΔX) to random numbers.

3. Calculate the fitness (Mean Squared Error MSE) values.

4. Start the static phase by updating the best fitness value and the best position found so far using a greedy selection.

5. If the fitness value was better than the best fitness so far, then update the food source

with the weight solution.

6. If the fitness value was worse than the worst fitness so far, then update the enemy source with the weight solution.

7. Start the dynamic phase by calculating the weights of separation, alignment, cohesion, attraction to food, distraction from enemy, and positions difference (s, a, c, f, e, w, and ΔX).

8. Calculate the separation, alignment, cohesion, attraction to food, and distraction from enemy values.

9. Update the dragonfly positions difference (ΔX).

10. Update the dragonfly positions (weights).

11. Repeat step 3-10 until maximum iteration is reached.

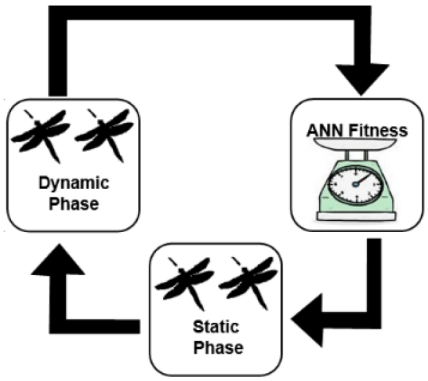

Figure 3.8: ANN-DA[13]

• **Case 2 : training and optimizing the ANN using the Grey Wolf Optimization (ANN-GWO)**

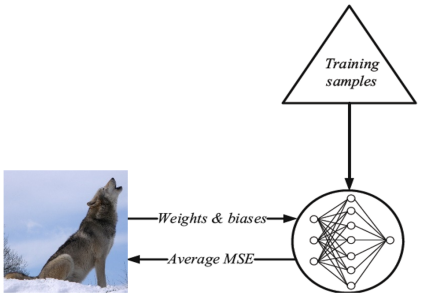

Figure 3.9: ANN-GWO[11]

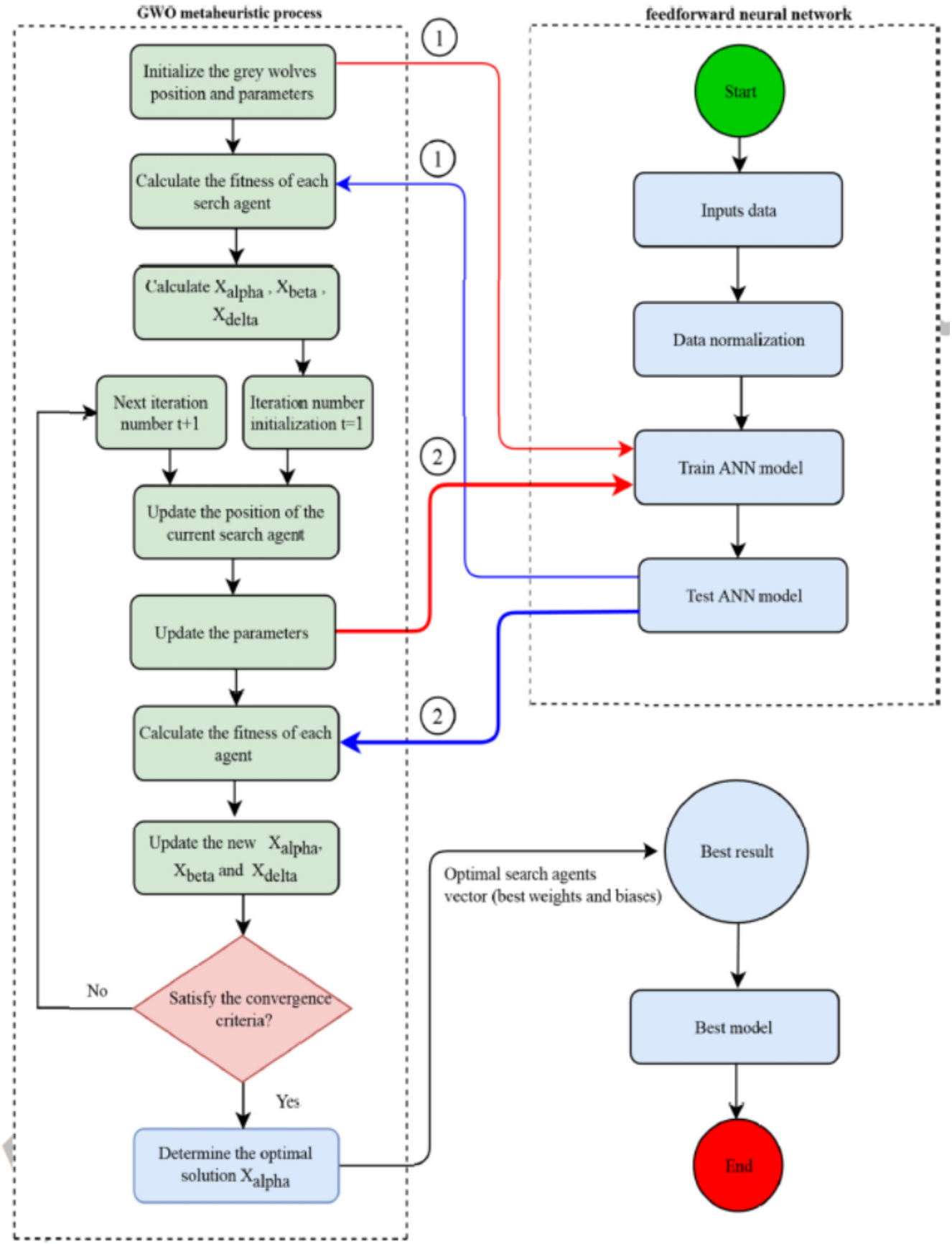

Figure 3.10: ANN-GWO execution steps[25]

Front-end

In our front-end system, The main part for the user-backend interaction, Giving the ability of exchanging data between user-backend, we gonna make simple access design for each part, and each request will interact with the backend related to. The Figure bellow will illustrate the whole mechanism.

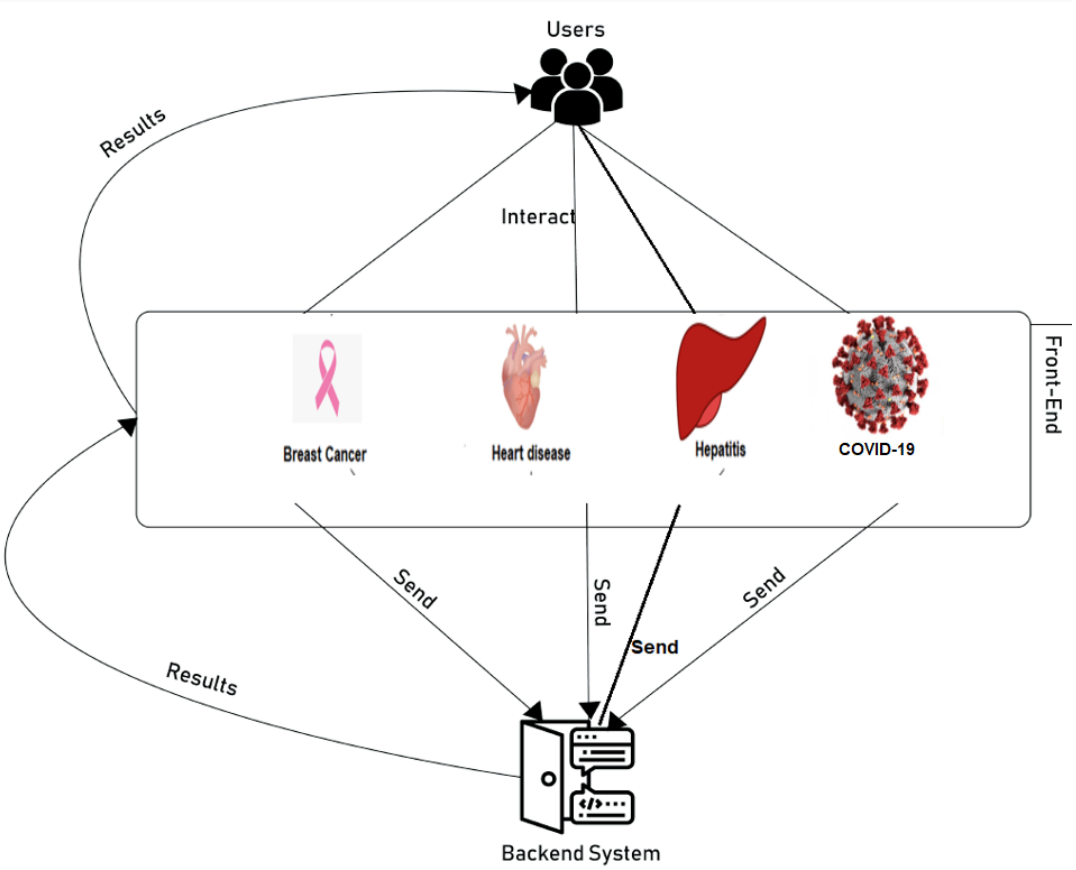

Figure 3.11: Front-end system

3.3 Summary

In this chapter we presented the general and detailed design of our system which consists of using nature based algorithms to train and optimize artificial neural networks from simple multilayer perceptrons MLP to deep convolutional neural networks for medical purposes.

We focused on using Dragonfly algorithm DA and Grey wolf optimization GWO as trainers and optimizers for artificial neural networks, in a technique called (ANN-DA) and (ANN-GWO), to build a prediction model for medical purposes.

Chapter 4

Implementation and Results

4.1 Environment and developing tools

To Develop our system, we are going to use matlab R2020a and its various tools, toolboxes and Add-Ons such as Simulink, Deep Learning Toolbox and ResNet-50 CNN Add-on. . . etc, that's for the backend part of the system, as for the front-end we are going to use Matlab App Designer which was first introduced in 2018, App Designer offers cutting edge design technology to build aesthetically pleasing user interfaces .

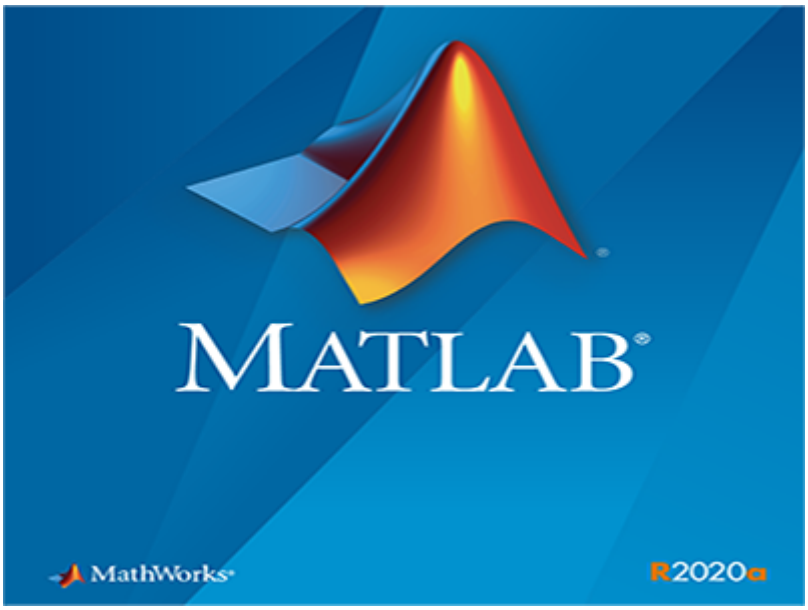

Figure 4.1: Matlab logo

4.2 Back-end Implementation

In this part of the project we are going to develop the backend system where the training and the processing happen using Matlab programming language and its various tools. Our project is devided into two parts , the first part is an exprimental comparison between training a Multilayer perceptron(MLP) using dragonfly algorithm(DA) and training a MLP using grey wolf optimization (GWO), and the second part is training/optimizing a convolutional neural network (CNN) using DA and comparing the results with other diffrent optimizer such as ADAM and SGDM.

4.2.1 Training/Optimizing MLP using DA or GWO

Loading/preprocessing datasets to train/optimize and test

Cancer dataset After searching and downloading the benchmark datasets, we want to load them into the memory so it can be pre-processed for training/optimizing and testing the MLP model.

```
function o=MLP_Cancer(solution)
load Cancer.txt
 x=Cancer;
 %I2=x(1:150,1:4);
 H2=x(1:699,2:11);
 for iii=1:699
     for jjj=1:10
         H2(iii,jjj)=((H2(iii,jjj)-1)/9);
     end
 end
 I2=H2(1:699,1:9);

 T=H2(1:699,10);
 Hno=19;
 dim=11*19;
        for ww=1:10*Hno
            W(ww)=solution(1,ww);
        end
        for bb=10*Hno+1:dim
            B(bb-(10*Hno))=solution(1,bb);
        end
        fitness=0;
        for pp=1:599
            actualvalue=my_simulate(9,Hno,1,W,B,I2(pp,:));
            fitness=fitness+(T(pp)-actualvalue)^2;
        end
        fitness=fitness/599;
        o=fitness;
end
```

Figure 4.2: loading and preprossesing the cancer dataset for training the MLP

```
    load Cancer.txt
    x=Cancer;
    %I2=x(1:150,1:4);
    H2=x(1:699,2:11);
    for iii=1:699
        for jjj=1:10
            H2(iii,jjj)=((H2(iii,jjj)-1)/9);
        end
    end
    I2=H2(1:699,1:9);

    T=H2(1:699,10);
    Hno=19;
    dim=11*19;
[Best_score,Best_pos,DA_cg_curve]=DA(SearchAgents_no,Max_iteration,lb,ub,dim,fobj,curve,app);
        Rrate=0;
        W=Best_pos(1:10*Hno);
        B=Best_pos(10*Hno+1:dim);
        for pp=600:699
            actualvalue=my_simulate(9,Hno,1,W,B,I2(pp,:) );
            if(T(pp)>=0.3 && T(pp)<0.4)
                if (abs(actualvalue-0.333333333333333)<0.1)
                    Rrate=Rrate+1;
                end
            end
            if(T(pp)>=0.1 && T(pp)<0.2)
                if (abs(actualvalue-0.111111111111111)<0.1)
                    Rrate=Rrate+1;
                end
            end
            app.AccuracyEditField.Value=strcat( num2str((Rrate/100)*100),'%');
            pause(0.02);
        end
        Final_rate=(Rrate/100)*100 ;
        app.AccuracyEditField.Value=strcat( num2str(Final_rate),'%');
        app.MSEEditField.Value=num2str(Best_score);
```

Figure 4.3: Optimizing using DA and testing the Trained MLP for breast cancer dataset

Heart disease dataset

```
function o=MLP_Heart(solution)
  load heart.txt
  x=heart;
  %I2=x(1:150,1:4);
  I2(:,1)=x(1:80,2);  I2(:,2)=x(1:80,3);  I2(:,3)=x(1:80,4);  I2(:,4)=x(1:80,5);
  I2(:,5)=x(1:80,6);  I2(:,6)=x(1:80,7);  I2(:,7)=x(1:80,8);  I2(:,8)=x(1:80,9);
  I2(:,9)=x(1:80,10); I2(:,10)=x(1:80,11);I2(:,11)=x(1:80,12);I2(:,12)=x(1:80,13);
  I2(:,13)=x(1:80,14);I2(:,14)=x(1:80,15);I2(:,15)=x(1:80,16);I2(:,16)=x(1:80,17);
  I2(:,17)=x(1:80,18);I2(:,18)=x(1:80,19);I2(:,19)=x(1:80,20);I2(:,20)=x(1:80,21);
  I2(:,21)=x(1:80,22);I2(:,22)=x(1:80,23);
  T=x(1:80,1);

  Hno=45;
 dim = 24*45+1;                      % Dimension of the problem

        for ww=1:23*Hno
            W(ww)=solution(1,ww);
        end
        for bb=23*Hno+1:dim
            B(bb-(23*Hno))=solution(1,bb);
        end
        fitness=0;
        for pp=1:80
            actualvalue=my_simulate(22,Hno,1,W,B,I2(pp,:) );

            fitness=fitness+(T(pp)-actualvalue)^2;

        end
        fitness=fitness/80;
        o=fitness;

end
```

Figure 4.4: loading and preprossesing the Heart disease dataset for training the MLP

```
load Heart_test.txt
   x=Heart_test;
   %I2=x(1:150,1:4);
   I2(:,1)=x(1:187,2);  I2(:,2)=x(1:187,3);  I2(:,3)=x(1:187,4);   I2(:,4)=x(1:187,5);
   I2(:,5)=x(1:187,6);  I2(:,6)=x(1:187,7);  I2(:,7)=x(1:187,8);   I2(:,8)=x(1:187,9);
   I2(:,9)=x(1:187,10); I2(:,10)=x(1:187,11);I2(:,11)=x(1:187,12); I2(:,12)=x(1:187,13);
   I2(:,13)=x(1:187,14);I2(:,14)=x(1:187,15);I2(:,15)=x(1:187,16); I2(:,16)=x(1:187,17);
   I2(:,17)=x(1:187,18);I2(:,18)=x(1:187,19);I2(:,19)=x(1:187,20); I2(:,20)=x(1:187,21);
   I2(:,21)=x(1:187,22);I2(:,22)=x(1:187,23);
   T=x(1:187,1);
   %I=(I2-0.1)./(7.9-0.1)
   Hno=45;
   dim = 24*45+1
[Best_score,Best_pos]=DA(sa,it,lb,ub,dim,fobj,curve,app);

       Rrate=0;
       W=Best_pos(1:23*Hno);
       B=Best_pos(23*Hno+1:dim);
       for pp=1:187
           actualvalue=my_simulate(22,Hno,1,W,B,I2(pp,:) );
           if(T(pp)==1)
               if (actualvalue>=0.95)
                   Rrate=Rrate+1;
               end
           end
           if(T(pp)==0)
               if (actualvalue(1)<0.05)
                   Rrate=Rrate+1;
               end
           end
           app.AccuracyEditField.Value=strcat( num2str((Rrate/187)*100),'%');
           pause(0.02);

       end

       Final_rate=(Rrate/187)*100;
       app.AccuracyEditField.Value=strcat( num2str(Final_rate),'%');
       app.MSEEditField.Value=num2str(Best_score);
```

Figure 4.5: Optimizing using DA and testing the trained MLP for heart disease dataset

Hepatitis dataset

```
function o=MLP_Hepatitis(solution)
   load hepatitis_2_csv.txt
   x=hepatitis_2_csv;
   %I2=x(1:150,1:4);
   I2(:,1)=x(1:568,2);  I2(:,2)=x(1:568,3);  I2(:,3)=x(1:568,4);  I2(:,4)=x(1:568,5);
   I2(:,5)=x(1:568,6);  I2(:,6)=x(1:568,7);  I2(:,7)=x(1:568,8);  I2(:,8)=x(1:568,9);
   I2(:,9)=x(1:568,10); I2(:,10)=x(1:568,11);I2(:,11)=x(1:568,12);I2(:,12)=x(1:568,13);
   I2(:,13)=x(1:568,14);I2(:,14)=x(1:568,15);I2(:,15)=x(1:568,16);I2(:,16)=x(1:568,17);
   I2(:,17)=x(1:568,18);I2(:,18)=x(1:568,19);I2(:,19)=x(1:568,20);I2(:,20)=x(1:568,21);
   I2(:,21)=x(1:568,22);I2(:,22)=x(1:568,23);
   T=x(1:568,1);

   Hno=45;
  dim = 24*45+1;                    % Dimension of the problem
       for ww=1:23*Hno
           W(ww)=solution(1,ww);
       end
       for bb=23*Hno+1:dim
           B(bb-(23*Hno))=solution(1,bb);
       end
       fitness=0;
       for pp=1:568
           actualvalue=my_simulate(22,Hno,1,W,B,I2(pp,:) );

           fitness=fitness+(T(pp)-actualvalue)^2;

       end
       fitness=fitness/568;
       o=fitness;

end
```

Figure 4.6: loading and preprossesing the Hepatitis dataset for training the MLP

```
 load test.txt
x=test;
%I2=x(1:150,1:4);
I2(:,1)=x(1:80,2);  I2(:,2)=x(1:80,3);  I2(:,3)=x(1:80,4);  I2(:,4)=x(1:80,5);
I2(:,5)=x(1:80,6);  I2(:,6)=x(1:80,7);  I2(:,7)=x(1:80,8);  I2(:,8)=x(1:80,9);
I2(:,9)=x(1:80,10); I2(:,10)=x(1:80,11);I2(:,11)=x(1:80,12);I2(:,12)=x(1:80,13);
I2(:,13)=x(1:80,14);I2(:,14)=x(1:80,15);I2(:,15)=x(1:80,16);I2(:,16)=x(1:80,17);
I2(:,17)=x(1:80,18);I2(:,18)=x(1:80,19);I2(:,19)=x(1:80,20);I2(:,20)=x(1:80,21);
I2(:,21)=x(1:80,22);
I2(:,22)=x(1:80,23);
T=x(1:80,1);
%I=(I2-0.1)./(7.9-0.1)
Hno=45;
dim = 24*45+1
    [Best_score,Best_pos]=DA(sahep,ithep,lb,ub,dim,fobj,curve,app);
    Rrate=0;
    W=Best_pos(1:23*Hno);
    B=Best_pos(23*Hno+1:dim);
    for pp=1:80
        actualvalue=my_simulate(22,Hno,1,W,B,I2(pp,:) );
        if(T(pp)==1)
            if (actualvalue>=0.95)
                Rrate=Rrate+1;
            end
        end
        if(T(pp)==0)
            if (actualvalue(1)<0.05)
                Rrate=Rrate+1;
            end
        end
        app.AccuracyEditField.Value=strcat( num2str((Rrate/80)*100),'%');
        pause(0.02);
    end
     Final_rate=(Rrate/80)*100;
    app.AccuracyEditField.Value=strcat( num2str(Final_rate),'%');
    app.MSEEditField.Value=num2str(Best_score);
```

Figure 4.7: Optimizing using DA and testing the Trained MLP for Hepatitis dataset

PS : in order to use GWO we just replace this line of code :

[Best_score,Best_pos,DA_cg_curve]=DA(SearchAgents_ no,Max_iteration,lb,ub,dim,fobj,curve,app);

with:

[Best_score,Best_pos,GWO_cg_curve]=GWO(SearchAgents_no,Max_iteration,lb,ub,dim,fobj,curve,app)

4.2.2 Trainig/optimizing CNN using DA for predicting COVID-19

Loading the dataset (X-ray images)

```
datapath='dataset';
% Image Datastore
imds=imageDatastore(datapath, 'IncludeSubfolders',true, 'LabelSource','foldernames');
% Determine the split up
total_split=countEachLabel(imds)
```

Figure 4.8: loading the dataset of X-ray images

Preprocessing the dataset

```
function Iout = preprocess_Xray(filename)

% Read the Filename
I = imread(filename);

% Some images might be RGB, convert them to Grayscale
if ~ismatrix(I)
    I=rgb2gray(I);
end

% Replicate the image 3 times to create an RGB image
Iout = cat(3,I,I,I);

end
```

Figure 4.9: Preprocessing function

Training the CNN

In this project we are working with Resnet50 CNN, we are training and optimizing this CNN using DA , and as a validation procedure we are implementing the K-fold cross validation(10 folds)

```
% Number of folds
num_folds=10;
% Loop for each fold
for fold_idx=1:num_folds
    fprintf('Processing %d among %d folds \n',fold_idx,num_folds);
   % Test Indices for current fold
    test_idx=fold_idx:num_folds:num_images;
    % Test cases for current fold
    imdsTest = subset(imds,test_idx);
    % Train indices for current fold
    train_idx=setdiff(1:length(imds.Files),test_idx);
    % Train cases for current fold
    imdsTrain = subset(imds,train_idx);
    % ResNet Architecture
    net=resnet50;
    lgraph = layerGraph(net);
    clear net;

    % Number of categories
    numClasses = numel(categories(imdsTrain.Labels));

```

Figure 4.10: setting our folds, CNN architacture, and classes

```
% Training Options, we choose a small mini-batch size due to limited images
options = trainingOptions('da',...
    'MaxEpochs',30,'MiniBatchSize',8,...
    'Shuffle','every-epoch', ...
    'InitialLearnRate',1e-4, ...
    'Verbose',false, ...
    'Plots','training-progress');

% Data Augumentation
augmenter = imageDataAugmenter( ...
    'RandRotation',[-5 5],'RandXReflection',1,...
    'RandYReflection',1,'RandXShear',[-0.05 0.05],'RandYShear',[-0.05 0.05]);

% Resizing all training images to [224 224] for ResNet architecture
auimds = augmentedImageDatastore([224 224],imdsTrain,'DataAugmentation',augmenter);

% Training
netTransfer = trainNetwork(auimds,lgraph,options);

% Resizing all testing images to [224 224] for ResNet architecture
augtestimds = augmentedImageDatastore([224 224],imdsTest);

% Testing and their corresponding Labels and Posterior for each Case
[predicted_labels(test_idx),posterior(test_idx,:)] = classify(netTransfer,augtestimds);

% Save the Independent ResNet Architectures obtained for each Fold
save(sprintf('ResNet50_%d_among_%d_folds',fold_idx,num_folds),'netTransfer','test_idx','train_idx');

% Clearing unnecessary variables
clearvars -except fold_idx num_folds num_images predicted_labels posterior imds netTransfer;
```

Figure 4.11: Training/Optimizing our cnn model using DA

4.3 Front-end

In the front-end part of the system, we want to create a simple design that can be easily used by diffrent demographics , but also aesthetically pleasing, that is why we are using Matlab APP designer, and the next figures will explain how our user interface works.

4.3.1 1st UI

we have 3 button :

• 1st button will open another UI where the user could apply experimental comparisons between DA and GWO by training/optimizing MLP's and with different datasets (breast cancer, heart disease and hepatitis).

• 2nd button will open a UI where the user browse for lungs X-ray image of a COVID-19 infected or non-infected specimen and predict the possibility of the invection, the user could also analyse the architecture of our CNN model (resnet50).

- 3rd button is the exit button.

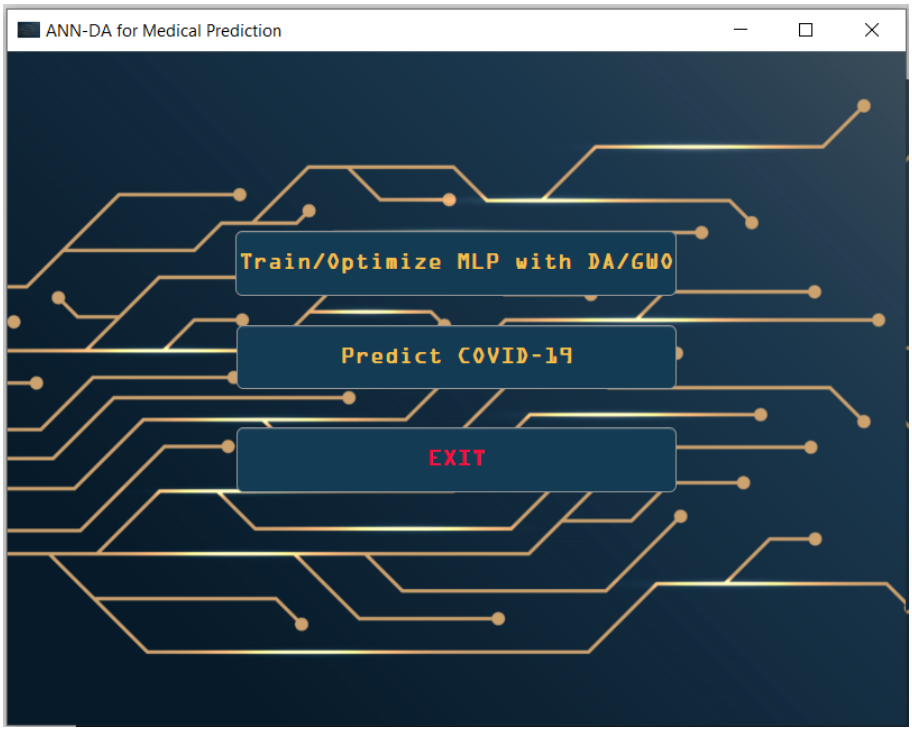

Figure 4.12: 1st UI

4.3.2 2nd UI

train/optimize MLP with DA/GWO :

- The user will first choose the dataset and the training algorithm (DA or GWO).
- Click train to train MLP
- Wait for resutls

The results are :

o Classification rate(accuracy).

o The Mean squared error (MSE).

o MSE curve that changes in real time.

o A text area where the MSE value changes in real time.

o Training time.

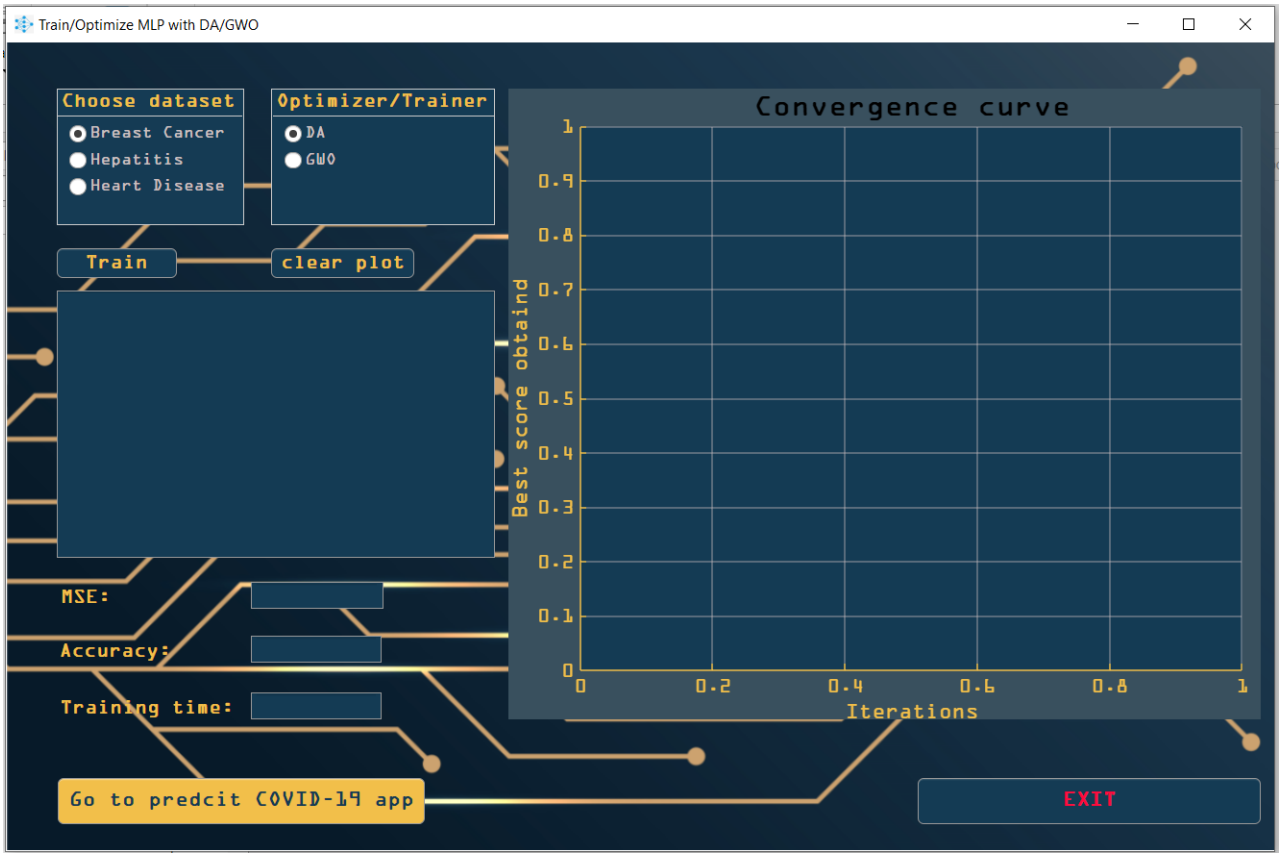

Figure 4.13: 2nd UI before results

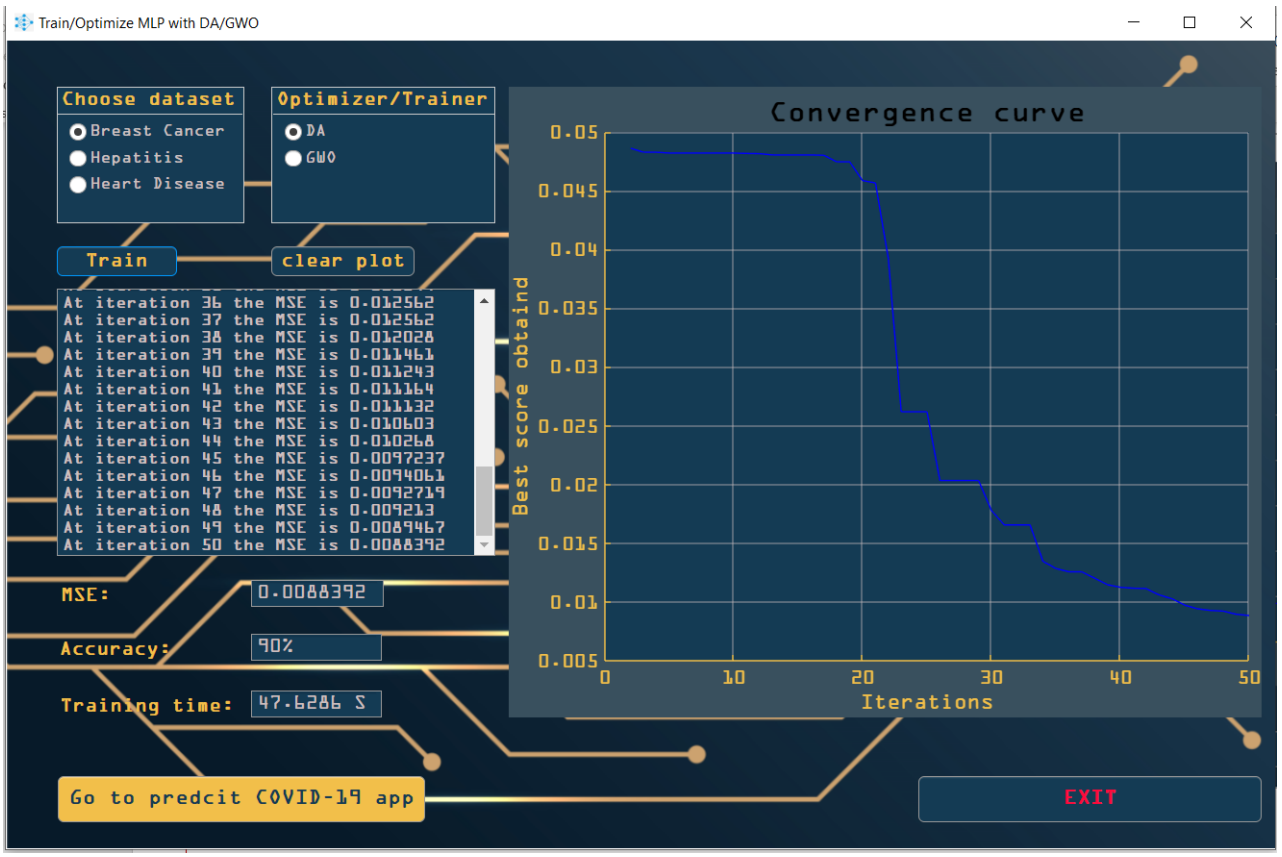

Figure 4.14: 2nd UI after results

4.3.3 3rd UI

predicting COVID-19 infection possibility.

The user browse for lungs X-ray image of a COVID-19 infected or non-infected specimen and predict the possibility of the infection, the user could also analyse the architecture of our CNN model (resnet50).

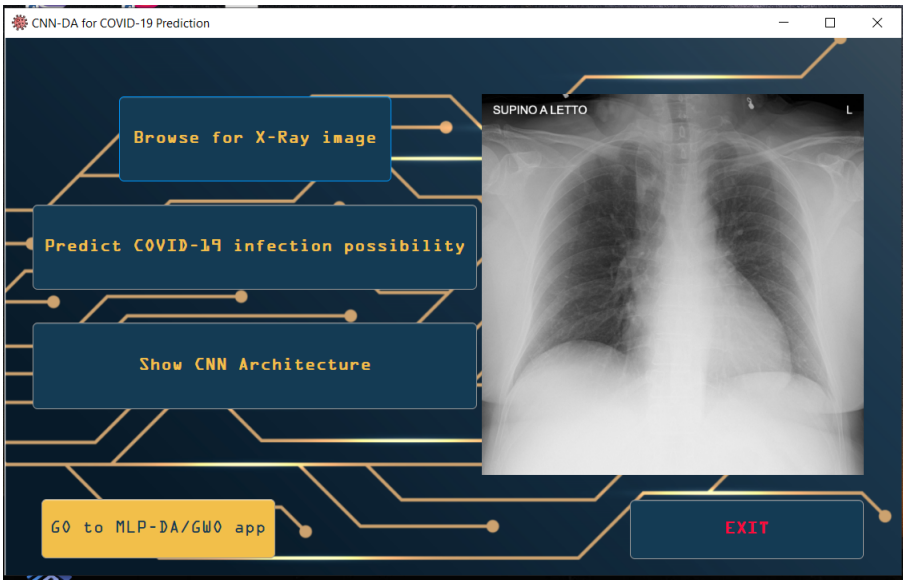

Figure 4.15: 3rd UI

- Predict COVID-19 button results :

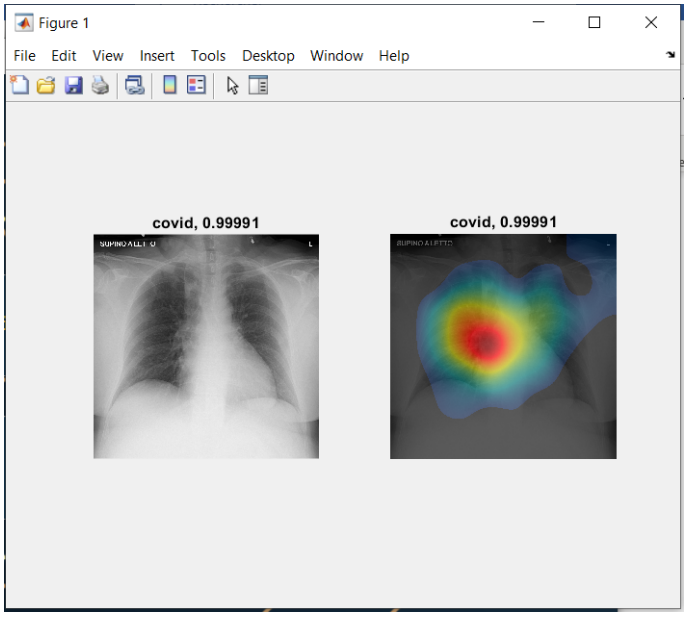

Figure 4.16: COVID-19 Prediction results

- Show CNN architecture button results :

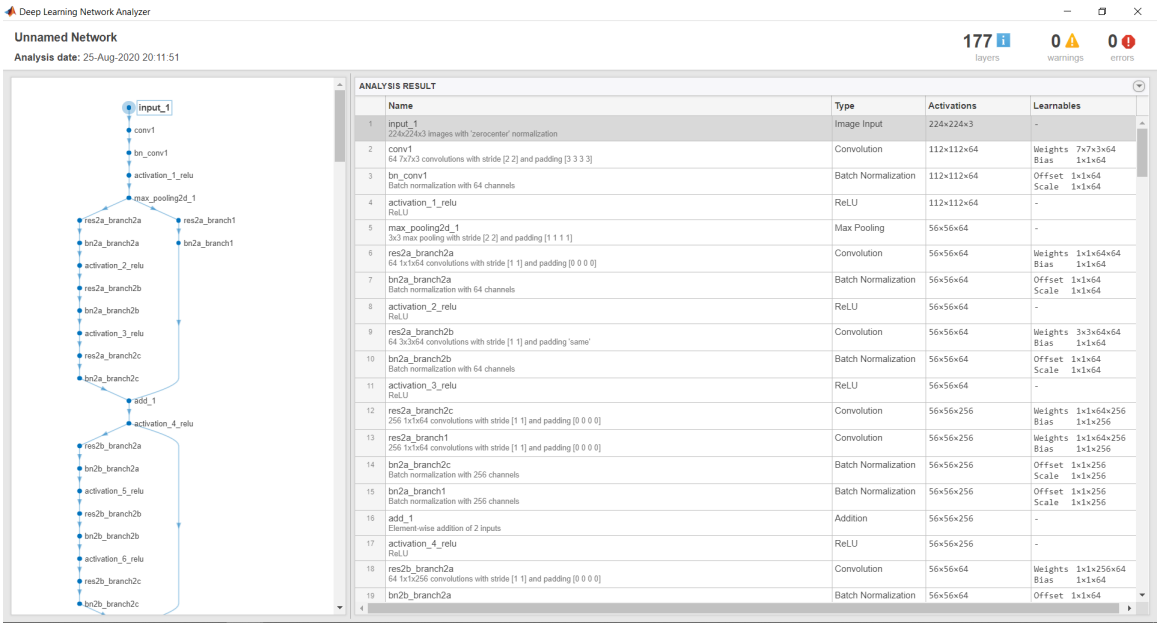

Figure 4.17: resNet50 CNN architecture.

NOTE: after training, our Resnet50 CNN has 177 layers, that is why we could not capture the full structure of the network, we discussed ResNet50 CNN in the 1st chapter for more details.

4.4 Expirements results

In this project we are using an Acer Aspire5 :

- Core i5 7th gen CPU.
- 8gigs of RAM.
- Nvidia Geforce MX940 GPU (2gigs of VRAM).
- 256gigs SSD for storage.
- Windows 10

4.4.1 MLP-DA and MLP-GWO comparison

Breast cancer dataset results

No.	MLP architecture	Search Agents	Iterations	Training time	Mean Squared Error(MSE)	Accuracy(%)
Test 1	MLP-DA	20	50	25.06 sec	0.0088	90%
Test 2	MLP-DA	200	200	40min7sec	0.0001	99%
Test 3	MLP-GWO	21	50	21.8834 sec	0.0061878	75%
Test 4	MLP-GWO	200	200	38min 47sec	0.00023	93%

Table 4.1: Breast cancer MLP training tests

Heart disease dataset results

No.	MLP architecture	Search Agents	Iterations	Training time	Mean Squared Error(MSE)	Accuracy(%)
Test 1	MLP-DA	20	50	30.9617sec	0.15713	80.2139%
Test 2	MLP-DA	65	200	13min	0.0432	90%
Test 3	MLP-GWO	20	50	14.6687sec	0.1131	66.3102%
Test 4	MLP-GWO	65	200	11min 23sec	0.0563	75.420%

Table 4.2: Heart disease MLP training tests

Hepatitis dataset results

No.	MLP architecture	Search Agents	Iterations	Training time	Mean Squared Error(MSE)	Accuracy(%)
Test 1	MLP-DA	20	50	43.8655sec	0.096658	80%
Test 2	MLP-DA	45	100	3min30sec	0.06596	82.5%
Test 3	MLP-GWO	20	50	14.6687sec	0.092654	65%
Test 4	MLP-GWO	45	100	112.2229sec	0.08289	65%

Table 4.3: Hepatitis MLP training tests

Discussion of the results of training MLPs

• From Table.1(breast cancer dataset) we can notice that DA performs better than GWO in training a multilayer perceptron (MLP) in terms of accuracy and MSE,although GWO is

slightly faster than DA.

• From Table 2 and 3 we can see that MLP-DA gives better accuracy and slightly higher MSE than MLP-GWO when the number of search agents and iteration is low, but MLP-GWO is still a little bit faster than MLP-DA because GWO has less number of phases than DA.

• On the long run we can conclude that if we add more iteration an search agents to MLP-DA it can be more effective and can out-perform MLP-GWO

4.4.2 CNN-DA and other CNN optimizers comparison (adam,sgdm. . . .) comparison

4.4.3 Simple CNN training tests

We are first going test different well-known optimizers such as (adam,sgdm) and DA with a simple CNN of 12 layers, here is its architecture :

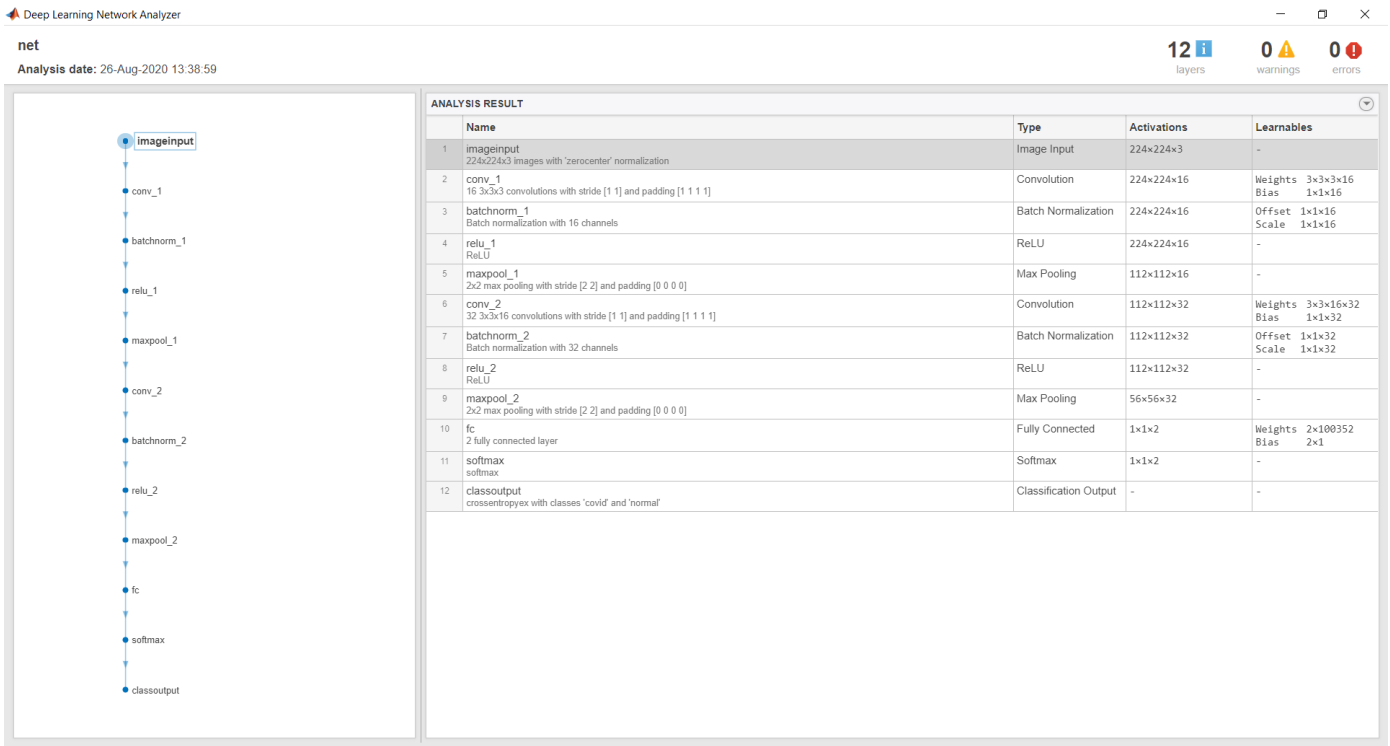

Figure 4.18: Simple CNN with 12 layers

No.	CNN architecture	Search agents	Iterations	Training time	Loss	Accuracy%
Test 1	CNN-sgdm	N/A	30	1min30sec	16.29%	85.71%
Test 2	CNN-adam	N/A	30	1min45sec	21.43%	78.57%
Test 3	CNN-DA	50	30	1min33sec	7.14%	92.86%

Table 4.4: simple CNN training tests

CNN-sgdm

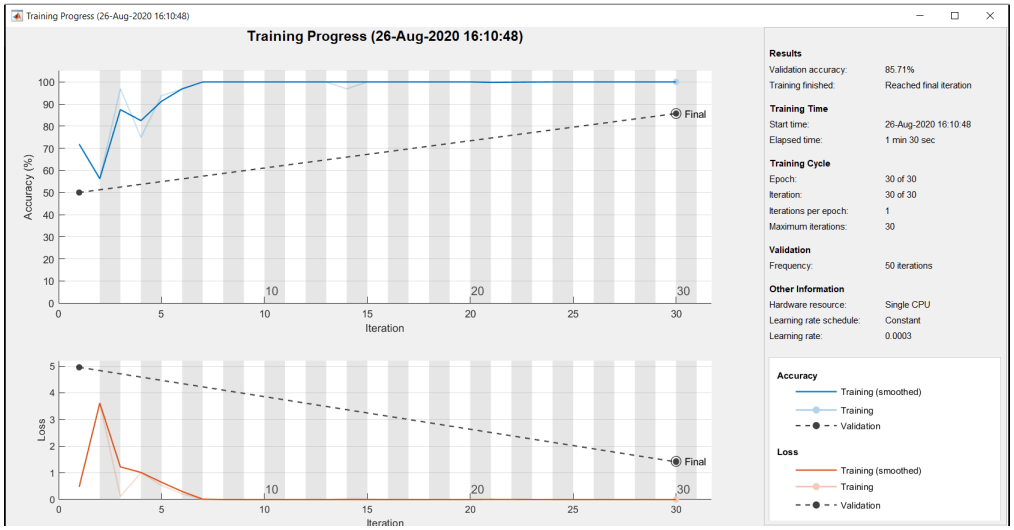

Figure 4.19: simple cnn trained by sgdm optimizer

CNN-adam

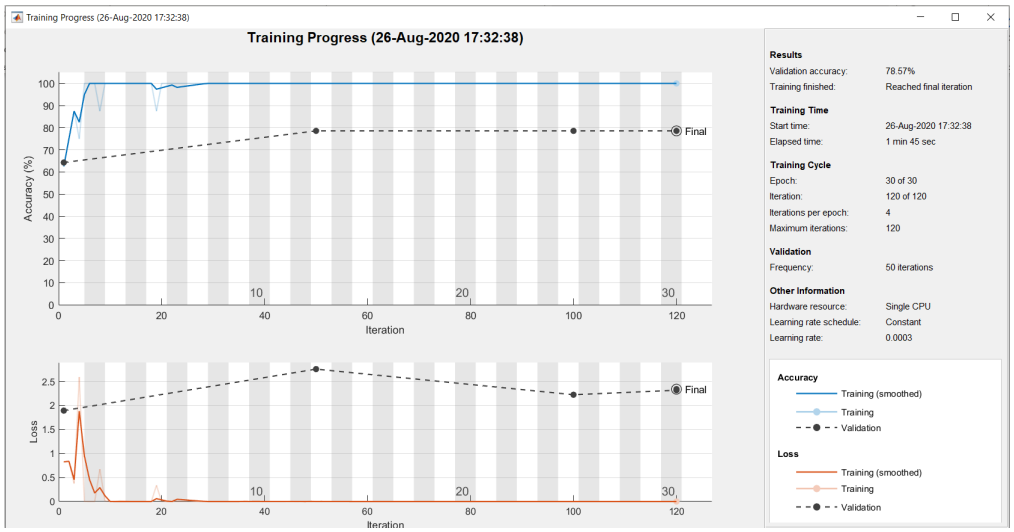

Figure 4.20: simple cnn trained by adam optimizer

CNN-DA

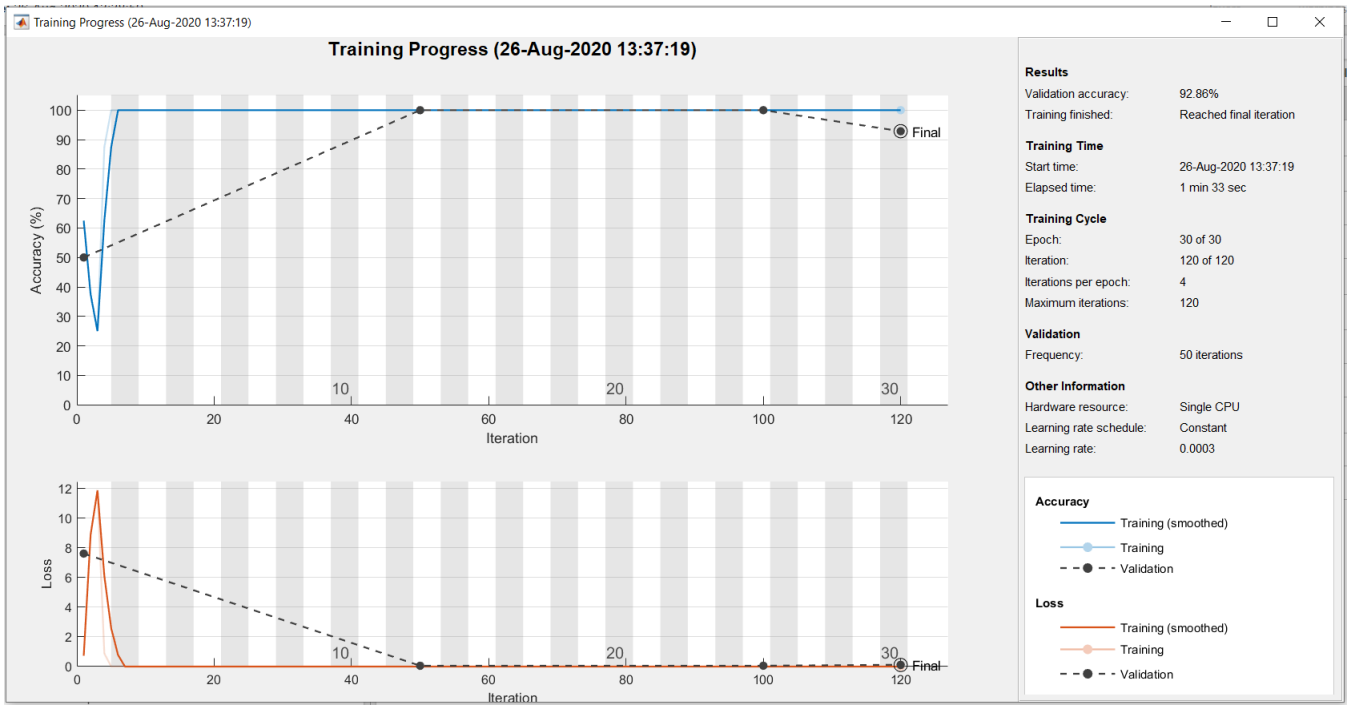

Figure 4.21: simple cnn trained by DA optimizer

4.4.4 Discussion of the results of training a simple CNN

• In terms of both accuracy and loss value and in a short period of time, we can obviously notice that even with a simple CNN architecture CNN-DA dominates in comparison with the CNN trained by ADAM optimizer or SGDM optimizer.

• This CNN architecture trained by dragonfly algorithm (CNN-DA) was able to achieve an accuracy of 92.86% in just 1min32sec which is a decent percentage knowing that this CNN has only 12 layers.

4.4.5 Resnet50 CNN-DA and Resnet50 CNN-ADAM comparison

No.	CNN architecture	Search agents	Iterations	Training time	Loss	Accuracy%
Test 1	resNet50-DA	50	300	2h20min29sec	2%	98%
Test 2	resNet50-SGDM	N/A	300	2h17min59sec	4%	96%

Table 4.5: resNet50 CNN training tests.

Resnet50 CNN-SGDM

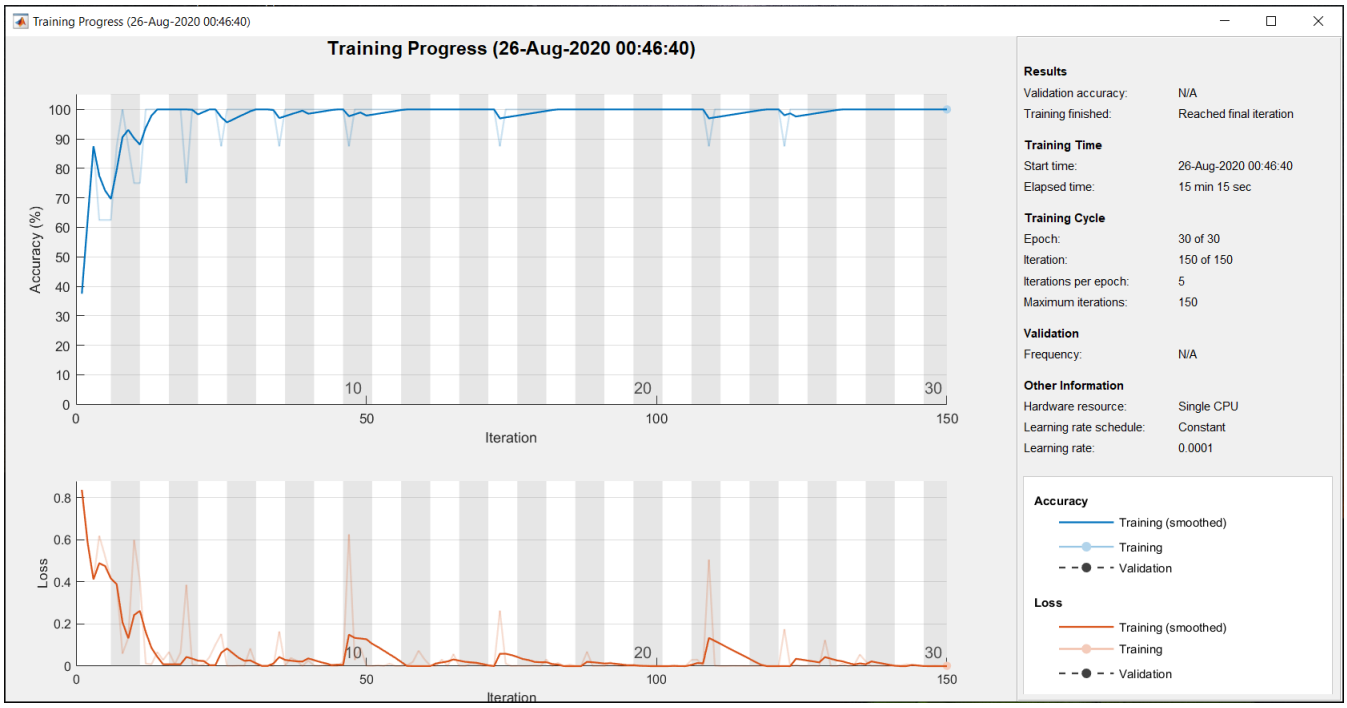

Figure 4.22: ResNet50 CNN trained by SGDM optimizer(10th fold)

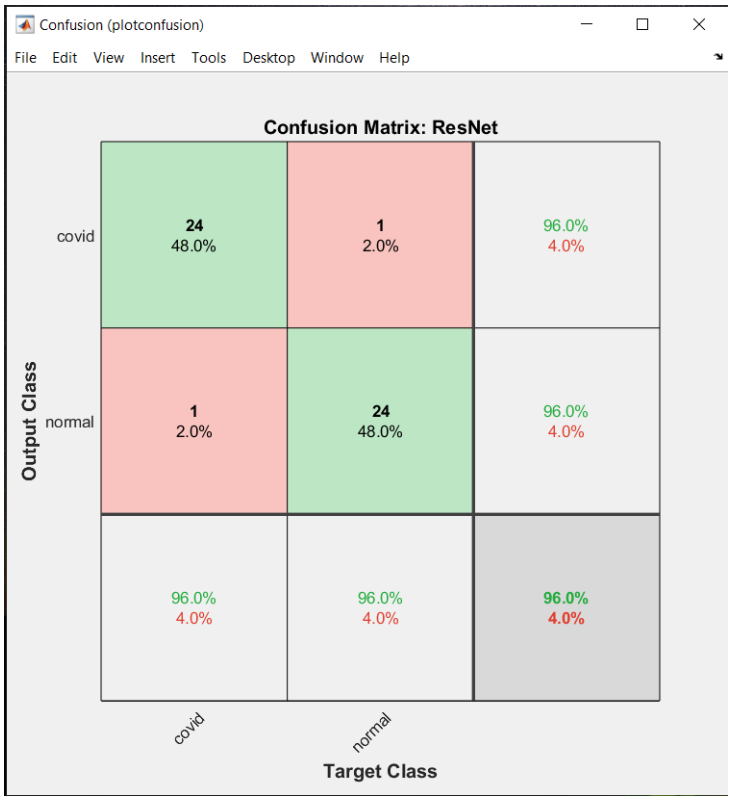

Figure 4.23: Confusion matrix of a trained resnet50 CNN-SGDM for predicting COVID-19

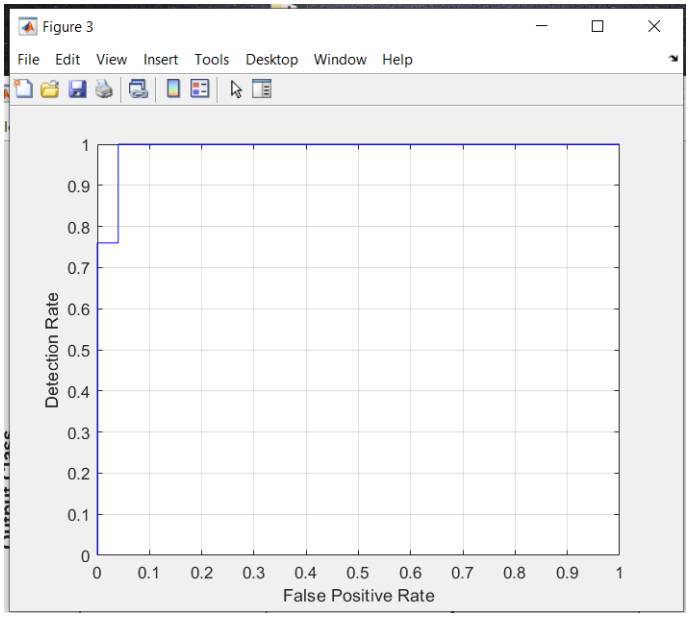

Figure 4.24: AUC(area under the curve) of the trained CNN-SGDM for predicting COVID-19 (0.9904)

ResNet50 CNN-DA

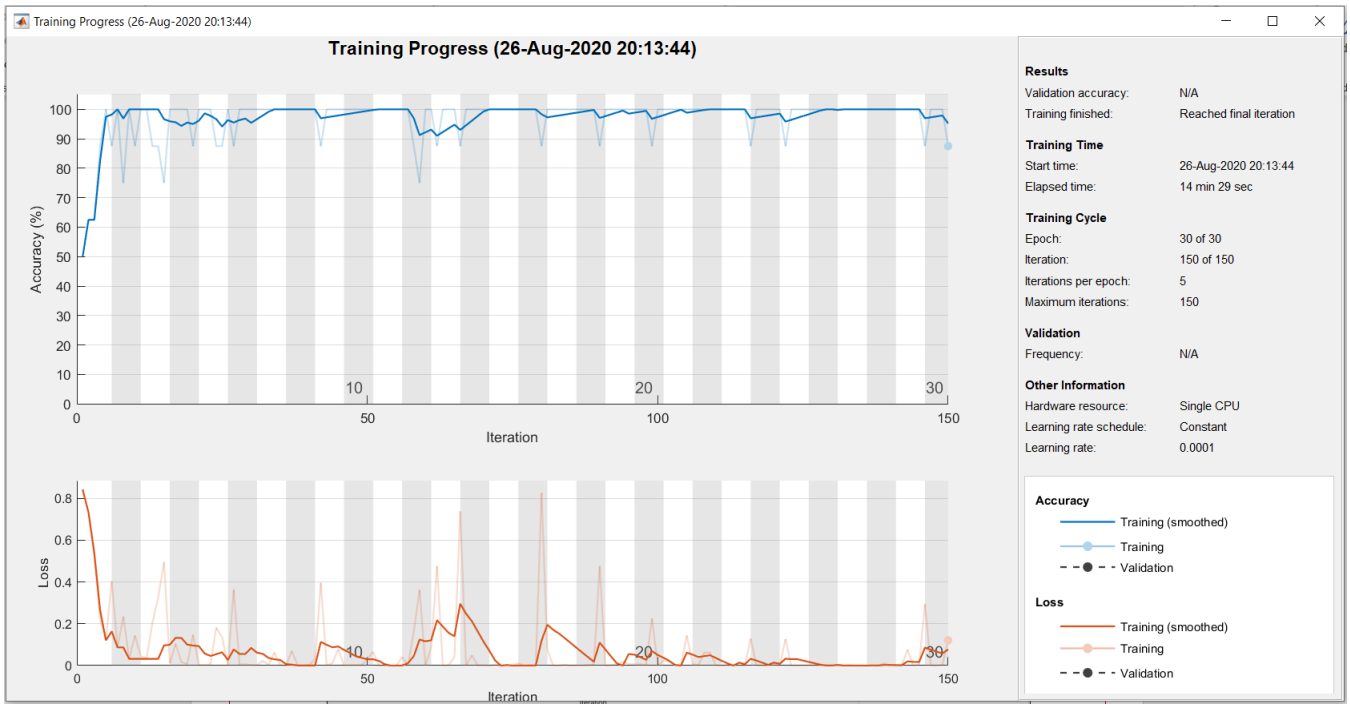

Figure 4.25: ResNet50 CNN trained by DA optimizer(10th fold)

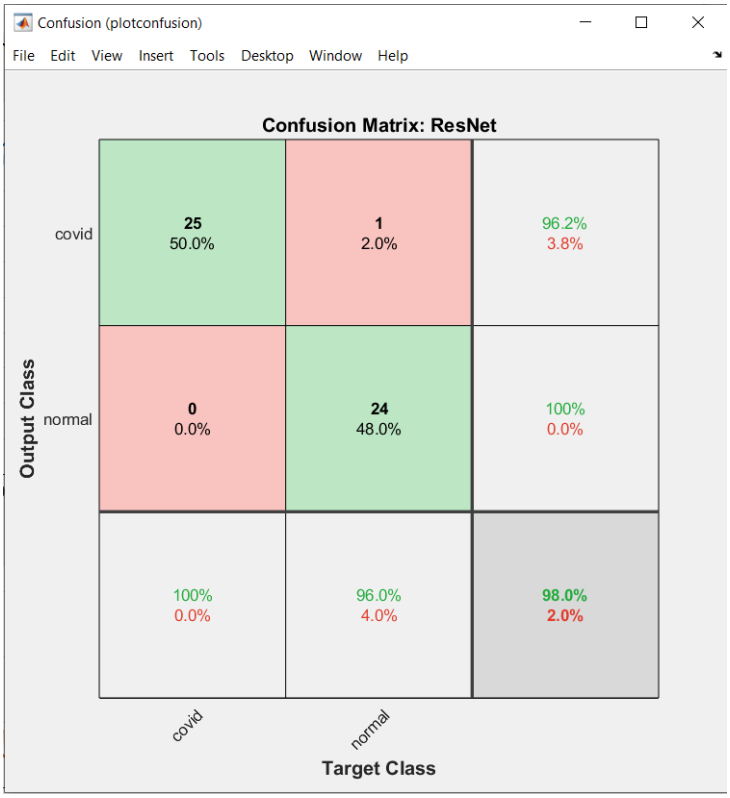

Figure 4.26: Confusion matrix of a trained resnet50 CNN-DA for predicting COVID-19

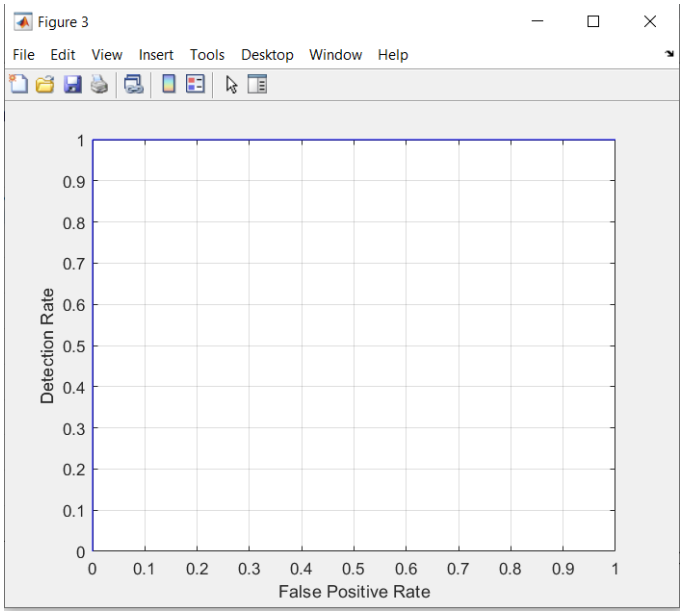

Figure 4.27: AUC(area under the curve) of the trained CNN-DA for predicting COVID-19 (1.000)

4.4.6 Discussion of results of training Resnet50 CNN

• Using deep learning Resnet50 CNN (see figure 52) we were able to build a COVID-19 prediction model, and after many tests using different optimizer/trainers such as ADAM and SGDM we come to the conclusion that Dragonfly algorithm DA is the more efficient and better performing algorithm in our case, which is image (X-Ray) classification for predicting the possibility of COVID-19 infection . • The results we got by training a ResNet50 CNN with SGDM optimizer were very promosing, we were able to 96% prediction accuracy in 2h17min59sec.

• We were able to reach an accuracy of 98% by training/optimizing ResNet50 with Dragonfly algorithm DA and using K-fold cross validation (10 folds) as a validation procedure

4.5 Limitaions

With the good results we had and achieved using the Deep learning technique (ResNet50 Convolution neural networks) to detect the possibility of COVID-19 infection from lungs X-Ray images, and during testing the performance of our system, here are the limitations we had:

• Our model cannot detect COVID-19 infection possibility from CT scan images or any ther forms of scanned images .

• Our model only works with X-Ray Images in order to detect COVID-19 infection possibility.

• Our model does not detect other lung infections meaning if the prediction is not covid-19(normal) does not mean the speciemen has healthy lungs.

4.6 Summary

The objective of this chapter was to present the general and detailed implementation of our system which consists of using nature based algorithms to train and optimize artificial neural networks from simple multilayer perceptrons MLP to deep convolutional neural networks for medical purposes.

In this project, we focused on using Dragonfly algorithm as trainer and optimizer for neural networks, and after many tests and comparisons with other optimizer such as Grey wolf optimization which is also nature based and also other well known optimizers in the deep learning realm such as ADAM and SGDM, DA proved to be very efficient, accurate and reliable when it comes to training and optimizing neural networks.

General Conclusion

AI applications can be highly beneficial for healthcare in general to the extent that it can revolutionise the entire healthcare system.

An AI system can have access to all this data, to make a quick diagnosis of conditions that require a more detailed examination. AI systems can use machine learning to correlate medical histories with various sensor readings, and learn to diagnose patients. With the records of millions of patients, maybe the system will be able to make even better predictions than the medical doctors. That would be one of the primary applications.

AI can also look at medical images, it could spot tumors where the physician might miss it, which has already been shown in AI literature.

This research goal was to use train artificial neural networks using Dragonfly Algorithm or ANN-DA (MLP-DA and CNN-da) to build a model that can predict diseases and compare its results in a comprehensive study with other trainer and optimizers such as (Grey Wolf Optimization ,ADAM, SGDM...etc) . To evaluate the proposed approaches, four medical real datasets have been used. Then, the data was divided into training sets and testing sets. The metrics taken into consideration in the results evaluation were accuracy, loss, MSE, AUC, and confusion matrix. Our models were tested using accuracy as a fitness function.

The results showed that ANN-DA got the highest accuracy in all of the datasets and got the highest AUC in all of the datasets.

Finally, based on the results we got, as a future work it is beneficial to test the proposed approaches on other problems, and compare it with other optimization algorithms. and improve our CNN model to predict whether the patient has COVID-19 or other anomolies or completely healthy lungs other than just COVID or NO-COVID.

Bibliography

[1] Pooja Kamat Amita Malav Kalyani Kadam. "PREDICTION OF HEART DISEASE USING K-MEANS and ARTIFICIAL NEURAL NETWORK as HYBRID APPROACH to IMPROVE ACCURACY". In: (2017).

[2] Sdiath Asiri. *Machine Learning Classifiers*. URL: https://towardsdatascience.com/machine-learning-classifiers-a5cc4e1b0623.

[3] Ishak barkat. "Character Recognition In The Image Using Deep Learning". 2019.

[4] Tyler Elliot Bettilyon. *introduction to deep learning*. URL: https://medium.com/tebs-lab/introduction-to-deep-learning-a46e92cb0022.

[5] Nina Danielsen. *Simple Image Classification with ResNet-50*. URL: https://medium.com/@nina95dan/simple-image-classification-with-resnet-50-334366e7311a.

[6] Giuseppe Pirlo Donato Impedovo. "eHealth and Artificial Intelligence". In: (2019).

[7] Bhumi Patel Sudipta Roy Himanshu KriplaniEmail. "Prediction of Chronic Kidney Diseases Using Deep Artificial Neural Network Techniqu". In: (2019).

[8] *Introduction to Reinforcement Learning*. URL: http://www.sra.vjti.info/blog/machine-learning/introduction-to-reinforcement-learning-in-2-minutes.

[9] Iztok Fister Iztok Fister jr Xin-She Yang. "A Brief Review of Nature-Inspired Algorithms for Optimization". In: (2013).

[10] Lukasz A. Kurgan Krzysztof J. Cios. *SPECT Heart Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/SPECT+Heart.

[11] Sayedli mirjalili. "How effective is the GreyWolf optimizer in training multi-layer perceptrons". In: (2015).

[12] Awhan Mohanty. *Multi layer Perceptron (MLP) Models on Real World Banking Data.* URL: https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f.

[13] Nadim Obeid Nailah Al-Madi Mais Yasen. "Optimizing Neural Networks using Dragonfly Algorithm for Medical Prediction". In: (2018).

[14] Sergio Nesmachnow. "An overview of metaheuristics: Accurate and efficient methods for optimisation". In: (2014).

[15] ayush pant. *Introduction to Machine Learning for Beginners.* URL: https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08.

[16] Adrian Rosebrock. *Detecting COVID-19 in X-ray images with Keras, TensorFlow, and Deep Learning.* URL: https://www.pyimagesearch.com/2020/03/16/detecting-covid-19-in-x-ray-images-with-keras-tensorflow-and-deep-learning/.

[17] Ankit Sachan. *Detailed Guide to Understand and Implement ResNets.* URL: https://cv-tricks.com/keras/understand-implement-resnets/.

[18] Mirjalili Sayedli. "Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems". In: (2015).

[19] Mirjalili Sayedli. *Grey Wolf Optimization.* URL: https://en.wikiversity.org/wiki/Algorithm_models/Grey_Wolf_Optimizer.

[20] Shdangwal. *A Brief and Simple Introduction to ResNet.* URL: https://medium.com/@shdangwal/a-brief-and-simple-introduction-to-resnet-47432eff95b8.

[21] Kumar Shridhar. *A Beginners Guide to Deep Learning.* URL: https://medium.com/botsupply/a-beginners-guide-to-deep-learning-5ee814cf7706.

[22] *Supervised vs. Unsupervised Machine Learning.* URL: https://medium.com/@chisoftware/supervised-vs-unsupervised-machine-learning-7f26118d5ee6.

[23] Chieh-ChenWu Wen-ChunYehb. "Prediction of fatty liver disease using machine learning algorithms". In: (2019).

[24] Dr. WIlliam H. Wolberg. *Breast Cancer Wisconsin (Original) Data Set.* URL: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+original.

[25] Tikhamarine Yazid and Malik Anurag. "Estimation of monthly reference evapotranspiration using novel hybrid machine learning approaches for solving single-objective, discrete, and multi-objective problems". In: (2019).

[26] yvkrishna. *ml-for-hepatitis-disease-prediction*. URL: `https://github.com/yvkrishna/ml-for-hepatitis-disease-prediction`.

[27] Olga Nikolayeva Zidian Xie. "Building Risk Prediction Models for Type 2 Diabetes Using Machine Learning Techniques". In: (2019).