DNN analysis using GA

Ansuman Sasmal
College of Engineering and Computer Science
Syracuse University
Syracuse, New York
asasmal@syr.edu

Bharat Jangama
College of Engineering and Computer Science
Syracuse University
Syracuse, New York
bjangama@syr.edu

***Abstract*—This paper presents an analysis of the performance of Deep Learning Neural Networks (DNN) using Genetic Algorithms (GA) in the context of Diabetic Retinopathy Debrecen Data Set. The goal of this study is to develop an accurate and efficient DNN model for the early detection of diabetic retinopathy, a common complication of diabetes that can lead to blindness.**

***Index Terms*—Deep Neural Network, Genetic Algorithms, optimization , analysis, time taken**

I. Introduction

In this paper, we present a study on the optimization of Deep Learning Neural Networks (DNN) using Genetic Algorithms (GA) for the analysis of the Diabetic Retinopathy Debrecen Data Set.

The GA-DNN hybrid model is used to optimize the DNN model's parameters, including the network algorithm, activation function for hidden neurons, number of hidden layers, and the number of neurons in each hidden layer.

To evaluate the performance of the GA-DNN model, various quality assessment criteria are used, including accuracy, precision, recall, and F1-score. These criteria are used to determine the effectiveness of the model's predictions in identifying the presence and severity of diabetic retinopathy.

The results of this study demonstrate the ability of the GA-DNN hybrid model to accurately and efficiently analyze the Diabetic Retinopathy Debrecen Data Set. The optimized DNN model achieved high accuracy, precision, recall, and F1-score, indicating its potential as a valuable tool for the early detection and diagnosis of diabetic retinopathy.

Overall, this study highlights the potential of GA-DNN models in medical diagnosis and disease detection, and has important implications for the development of more accurate and efficient DNN models in healthcare applications.

II. Genetic Algorithms

Charles Darwin's idea of natural evolution served as the foundation for the search heuristic known as the genetic algorithm. The fittest people are chosen for reproduction in order to give rise to the next generation's children, which is how natural selection works [2].

The genetic algorithm (GA), developed by John Holland and his collaborators in the 1960s and 1970s (Holland, 1975; De Jong, 1975), is a model or abstraction of biological evolution based on Charles Darwin's theory of natural selection. Holland was probably the first to use the crossover and recombination, mutation, and selection in the study of adaptive and artificial systems. These genetic operators form the essential part of the genetic algorithm as a problem-solving strategy. Since then, many variants of genetic algorithms have been developed and applied to a wide range of optimization problems, from graph coloring to pattern recognition, from discrete systems (such as the travelling salesman problem) to continuous systems (e.g., the efficient design of airfoil in aerospace engineering), and from financial markets to multi-objective engineering optimization [3].

There are many advantages of genetic algorithms over traditional optimization algorithms. Two most notable are: the ability of dealing with complex problems and parallelism. Genetic algorithms can deal with various types of optimization, whether the objective (fitness) function is stationary or non-stationary (change with time), linear or nonlinear, continuous or discontinuous, or with random noise. Because multiple offsprings in a population act like independent agents, the population (or any subgroup) can explore the search space in many directions simultaneously. This feature makes it ideal to parallelize the algorithms for implementation. Different parameters and even different groups of encoded strings can be manipulated at the same time.

However, genetic algorithms also have some disadvantages. The formulation of fitness function, the use of population size, the choice of the important parameters such as the rate of mutation and crossover, and the selection criteria of the new population should be carried out carefully. Any inappropriate choice will make it difficult for the algorithm to converge or it will simply produce meaningless results. Despite these drawbacks, genetic algorithms remain one of the most widely used optimization algorithms in modern nonlinear optimization.

The optimal solution for a problem can be achieved through a series of steps that include encoding the objectives or cost functions, defining a fitness function, creating a population of individuals, evaluating their fitness, performing crossover and mutation, and replacing the old population with a new one.

One iteration of this process is called a generation, and during each generation, fixed-length character strings are used. However, variable-length strings and coding structures have also been researched extensively. The objective function is typically encoded using binary or real-valued arrays, with binary strings being the most commonly used form of encoding.

Crossover and mutation are the two main genetic operators used in GA. Crossover is the process of swapping a segment of one chromosome with the corresponding segment on another chromosome at a random position. Single-point crossover is the most commonly used form of crossover, but multiple-point crossover can also be used to increase the efficiency of the algorithm. Mutation is achieved by flipping randomly selected bits in the chromosome. The probability of mutation is usually small, and mutation can occur at multiple sites simultaneously.

Algorithm 1 Genetic Algorithm

```
Objective function $\longrightarrow f(x)$, $x = (x_1, ..., x_d)^T$

Encode the solutions into chromosomes (strings)
Define fitness $\longrightarrow F$ (e.g., $F \propto f(x)$ for maximization)
Generate the initial population
Initialize the probabilities of crossover $\longrightarrow (p_c)$ and mutation $(p_m)$

while $t < Max\ number\ of\ generations$ do
    Generate new solution by crossover and mutation
    Crossover with a crossover probability $p_c$
    Mutate with a mutation probability $p_m$
    Accept the new solutions if their fitness increases
    Select the current best for the next generation (elitism)
    Update $t \longrightarrow t + 1$
end while

Decode the results and visualization
```

Pseudocode of the algorithm:

- Objective function f(x), x = (x1, ..., xd)T: This is the function that you are trying to optimize. It takes a vector of d variables x as input and returns a scalar value f(x) that represents the quality of the solution.
- Encode the solutions into chromosomes (strings): In order to apply genetic operations such as crossover and mutation, we need to encode the solutions into a form that can be manipulated as strings of bits or characters.
- Define fitness F (eg, F f(x) for maximization): The fitness function is a mapping from the solutions (chromosomes) to their fitness values. In this case, we define the fitness to be proportional to the objective function f(x) for maximization problems. For minimization problems, we would take the negative of f(x).
- Generate the initial population: This is the starting point of the algorithm. We randomly generate a set of solutions (chromosomes) to form the initial population.
- Initialize the probabilities of crossover (pc) and mutation (pm): These are parameters of the algorithm that control the probability of applying the genetic operations of crossover and mutation.
- while (t ¡Max number of generations): This is the main loop of the algorithm. We repeat the following steps until we reach the maximum number of generations or until some convergence criterion is met.
- Generate new solution by crossover and mutation: We create new solutions by applying genetic operations such as crossover and mutation to the current population.
- Crossover with a crossover probability pc: This is a genetic operator that combines two parent solutions to produce a new offspring solution. The crossover probability pc controls the likelihood of applying this operation.
- Mutate with a mutation probability pm: This is a genetic operator that introduces small random changes to a solution to explore new regions of the search space. The mutation probability pm controls the likelihood of applying this operation.
- Accept the new solutions if their fitness increase: We evaluate the fitness of the new solutions and accept them if their fitness is better than the parent solutions.
- Select the current best for the next generation (elitism): We select the best solutions (chromosomes) from the current population to form the next generation. This is known as elitism and ensures that we do not lose the best solutions found so far.
- Update t = t + 1: We increment the generation counter and repeat the loop.
- Decode the results and visualization: Once the algorithm has terminated, we decode the solutions (chromosomes) back into their original form (vector of variables x) and visualize the results.

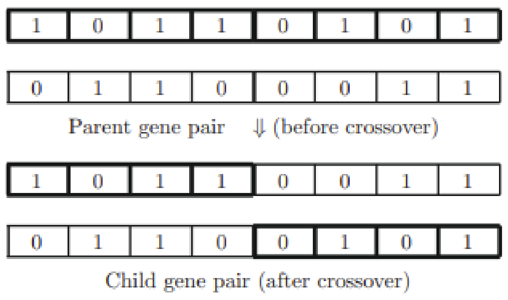
1 0 1 1 0 1 0 1
0 1 1 0 0 0 1 1
Parent gene pair ⇓ (before crossover)
1 0 1 1 0 0 1 1
0 1 1 0 0 1 0 1
Child gene pair (after crossover)

Fig. 1. Diagram of crossover at a random crossover point (location) in genetic algorithms

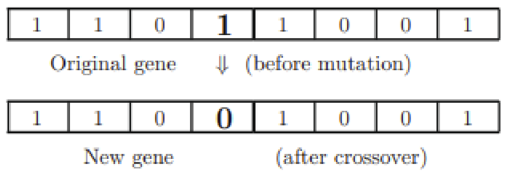

Fig. 2. Schematic representation of mutation at a single site by flipping a randomly selected bit (1 $\to$ 0).

III. Applications of Genetic Algorithms

Genetic algorithms have a wide range of applications in various fields, some of which include:

- Optimization Problems: Genetic algorithms can be used to solve complex optimization problems, such as maximizing profits or minimizing costs in businesses, optimizing the design of engineering systems, and finding optimal solutions in logistics and transportation [4].
- Machine Learning: Genetic algorithms can be used in machine learning tasks, such as feature selection, parameter tuning, and neural network architecture optimization [5].
- Robotics: Genetic algorithms can be used to optimize robot motion planning and control, and to design robot configurations for specific tasks [6].
- Game Development: Genetic algorithms can be used to develop artificial intelligence (AI) for games, including optimizing game strategies and generating game content [7].
- Image and Signal Processing: Genetic algorithms can be used for feature extraction and feature selection in image and signal processing applications, such as speech recognition, image classification, and video compression [8].
- Financial Modeling: Genetic algorithms can be used to optimize portfolio management, forecast stock prices, and detect anomalies in financial data [10].
- Bioinformatics: Genetic algorithms can be used for protein structure prediction, drug design, and DNA sequencing [9].

IV. Experiment Setup

Diabetic Retinopathy (DR) is a leading cause of blindness worldwide. Early detection and diagnosis of DR are crucial to prevent vision loss. Machine learning techniques, including neural networks, have shown promise in detecting DR from medical images. In this report, we describe our attempts to develop a neural network architecture for DR diagnosis using a Genetic Algorithm (GA) to optimize the architecture.

We used the Diabetic Retinopathy Debrecen Data Set, which contains features extracted from the Messidor image set to predict whether an image contains signs of DR or not. The dataset contains a total of 1151 instances, each of which represents an eye from a patient with diabetes [1].

We generated six models and made analysis to understand which performed better.Each model has a different configuration, mutation rate, crossover rate, population size, epsilon, and iteration. The table summarizes the accuracy of each model, the time it took to train, and the confusion matrix for each model.

- Model 1: This has a neural network configuration of (19,38)(38,19)(19,1) with roulette selection, gaussian mutation, and uniform crossover. It has a high mutation rate of 0.95 and a crossover rate of 0.9. The population size is 2x19x39, and the stopping criterion is when the accuracy reaches a certain threshold or after 200 iterations. The accuracy achieved by Model 1 is 52.3 percent, which is the lowest among all models. It took 40 minutes to train the model. The confusion matrix shows that the model has poor performance in predicting the labels, as the values are low and unbalanced.
- Model 2: This has a more complex neural network configuration of (19,128), (128,256), (256,512), (512,256), (256,128),(128,1) with roulette selection, gaussian mutation, and uniform crossover. It has a mutation rate of 0.95 and a crossover rate of 0.9. The population size is 2x19x129, and the stopping criterion is after 40 iterations. Model 2 achieved an accuracy of 60.61 percent, which is a significant improvement over Model 1. It took 10.5 hours to train the model. The confusion matrix shows that the model has better performance in predicting the labels, with more balanced and higher values.
- Model 3: This has a neural network configuration of (19,128), (128,256), (256,128), (128,1) with roulette selection, gaussian mutation, and uniform crossover. It has a mutation rate of 0.95 and a crossover rate of 0.9. The population size is 2x19x129, and the stopping criterion is after 40 iterations. Model 3 achieved an accuracy of 60.03 percent, which is slightly lower than Model 2. It took 4.4 hours to train the model. The confusion matrix shows that the model has good performance in predicting the labels, with balanced and high values.
- Model 4: This has a neural network configuration of (19,128), (128,256), (256,128), (128,1) with rouletteinv selection, gaussian mutation, and uniform crossover. It has a mutation rate of 0.95 and a crossover rate of 0.9. The population size is 2x19x129, and the stopping criterion is after 40 iterations. Model 4 achieved an accuracy of 60.18 percent, which is slightly higher than Model 3. It took 3.8 hours to train the model. The confusion matrix shows that the model has good performance in predicting the labels, with balanced and high values.
- Model 5: This Model has an NN configuration of (19,128), (128,256), (256,128), (128,1) and uses the roulette selection method, uniform crossover, and Gaussian mutation with a rate of 0.95. The crossover rate and epsilon value are set to 0.9 and 0.02, respectively. The model is run for 100 iterations with a population size of 2x19x129, and it achieves an accuracy of 61.73 percent with a time of 5.3 hours. The confusion matrix of this model shows that it correctly identifies 85.0 percent of class 0 samples and 77.0 percent of class 1 samples.
- Model 6: This Model has an NN configuration of (19,128), (128,256), (256,128), (128,1) and uses the rouletteinv selection method, uniform crossover, and Gaussian mutation with a rate of 0.95. The crossover rate and epsilon value are set to 0.9 and 0.02, respectively. The model is run for 30 iterations with a population size of 2x19x129, and it achieves an accuracy of 62.9 percent with a time of 3.9 hours. The confusion matrix of this model shows that it correctly identifies 97.6 percent of class 0 samples and 64.4 percent of class 1 samples.

TABLE I
MODEL ANALYSIS

Model	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
NN config	(19,38),(38,19), (19,1)	(19,128),(128,256), (256,512), (512,256), (256,128),(128,1)	(19,128),(128,256), (256,128), (128,1)	(19,128),(128,256), (256,128), (128,1)	(19,128),(128,256), (256,128), (128,1)	(19,128),(128,256), (256,128), (128,1)
selection	roulette	roulette	roulette	rouletteinv	roulette	rouletteinv
Mutation	gaussian(0.1)	gaussian(0.1)	gaussian(0.1)	gaussian(0.1)	gaussian(0.1)	gaussian(0.1)
crossover	uniform crossover	uniform crossover	uniform crossover	uniform crossover	uniform crossover	uniform crossover
mutation rate	0.95	0.95	0.95	0.95	0.95	0.95
crossover rate	0.9	0.9	0.9	0.9	0.9	0.9
population size	2*19*39	2*19*129	2*19*129	2*19*129	2*19*129	2*19*129
epsilon	0.02	0.02	0.02	0.02	0.02	0.02
iteration	200	40	40	40	100	30
accuracy	52.3	60.61	60.03	60.18	61.73	62.9
time	0.66 hrs	10.5 hrs	4.4 hrs	3.8 hrs	5.3 hrs	3.9 hrs
confusion matrix	61.8 100.2 57.4 104.6	79.6 82.4 78.3 83.7	72.1 89.9 67.8 94.2	79.3 82.7 84.4 77.6	85.0 77.0 89.6 72.4	97.6 64.4 93.0 69.0

V. OBSERVATIONS

A. From the analysis

From the above observations, we can see that Model 6 achieves the highest accuracy among all models with a value of 62.9 percent. However, its performance on class 1 samples is relatively lower than that of other models. Model 5 also performs well with an accuracy of 61.73 percent and better performance on class 1 samples than Model 6. The rest of the models have lower accuracy and performance compared to Models 5 and 6. The choice of the appropriate model for this classification task would depend on the specific requirements and constraints of the application.

B. From the graph

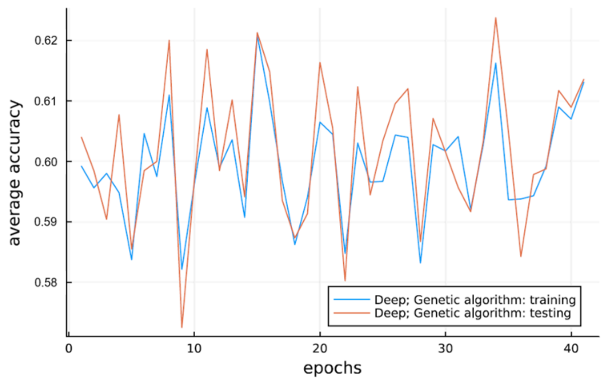

Fig. 4. Model 2

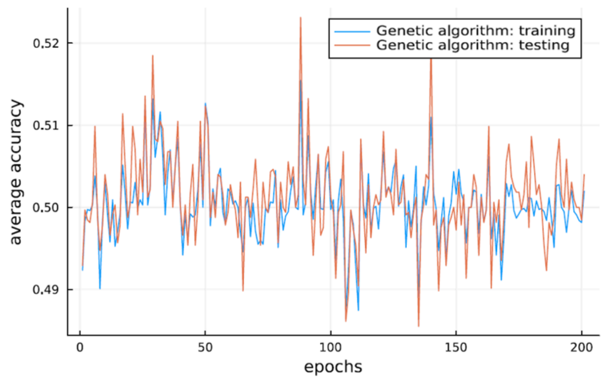

Fig. 3. Model 1

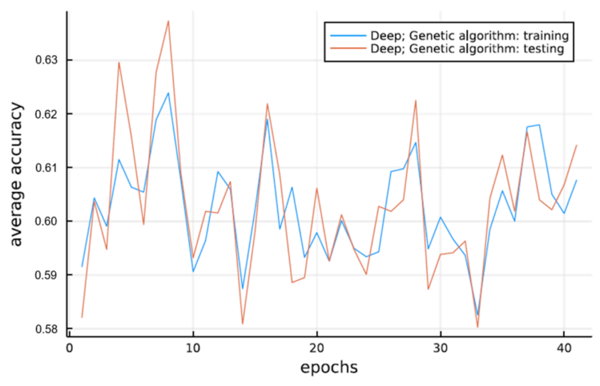

Fig. 5. Model 3

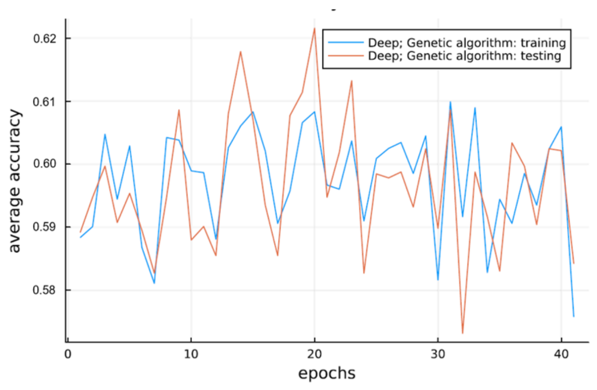

Fig. 6. Model 4

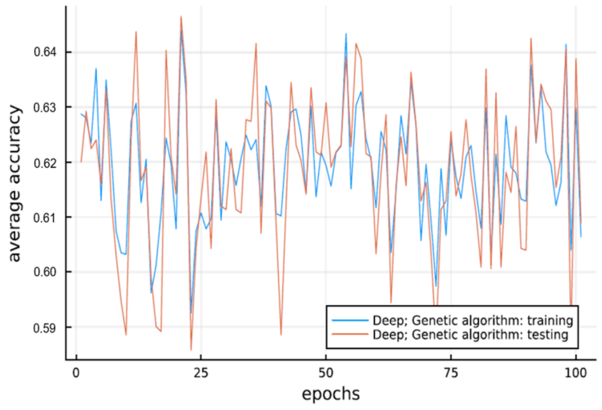

Fig. 7. Model 5

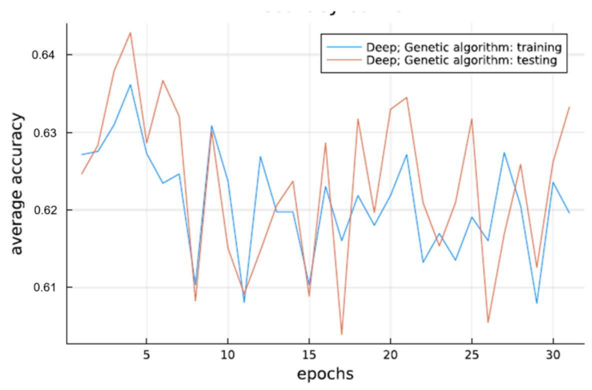

Fig. 8. Model 6

From images Figure 3 Figure 4 Figure 5 Figure 6 Figure 7 Figure 8, we see a lot of volatility in accuracy with each iteration, and on calculating the mean accuracy of all the iterations achieved we come to the conclusion that model 6 performed the best compared to others in terms of accuracy.

C. *From the confusion matrix*

After generating a confusion matrix we see that Model 6 Figure 14 performed the best compared to others.

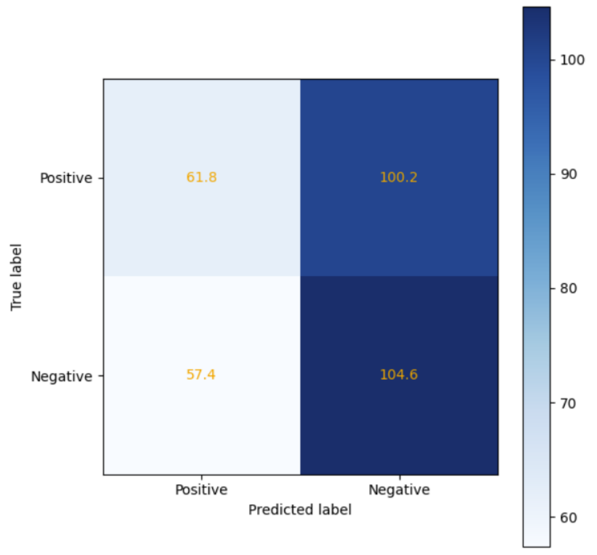

Fig. 9. Confusion Matrix Model 1

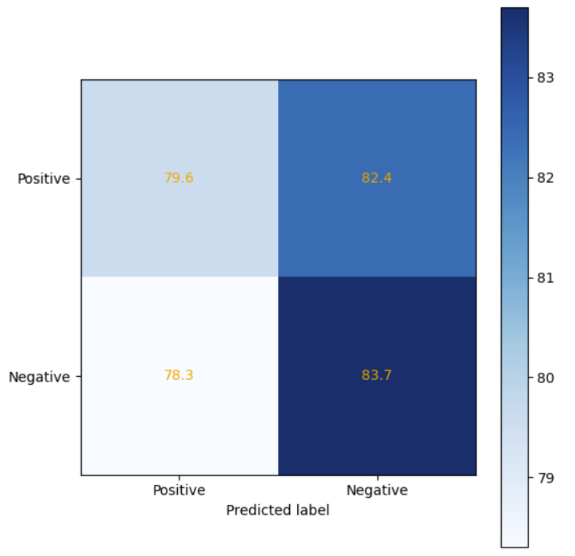

Fig. 10. Confusion Matrix Model 2

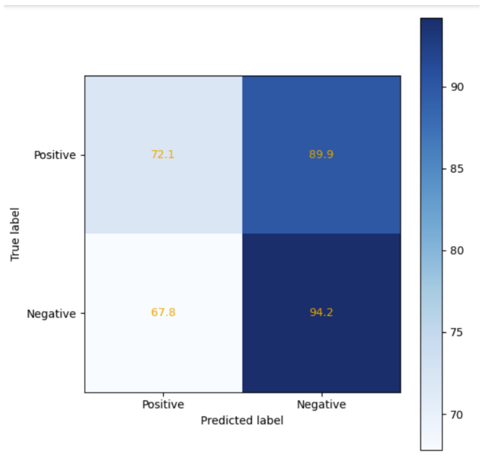

Fig. 11. Confusion Matrix Model 3

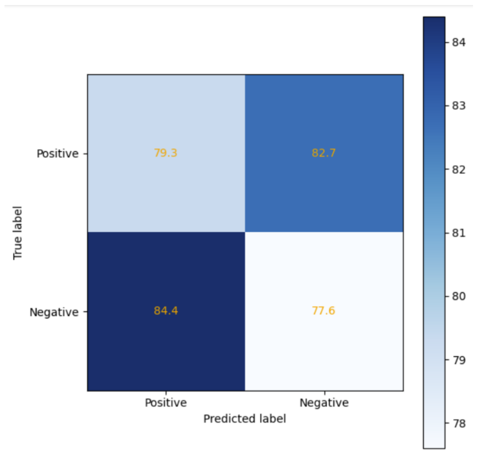

Fig. 12. Confusion Matrix Model 4

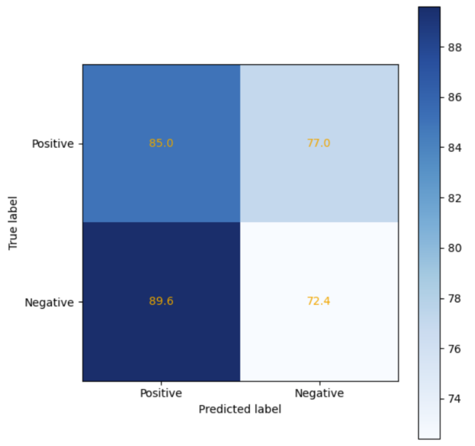

Fig. 13. Confusion Matrix Model 5

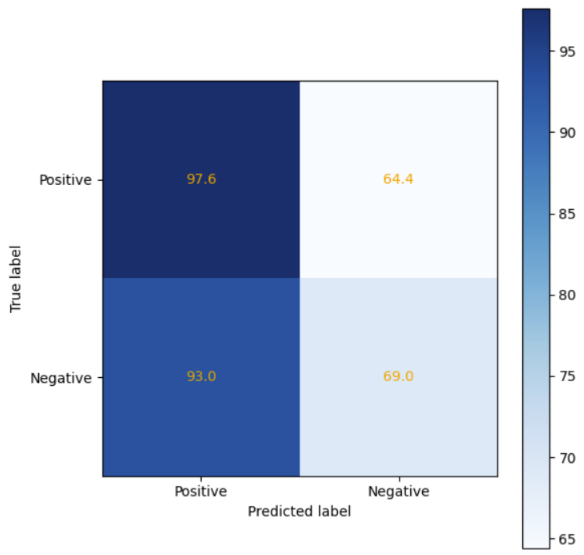

Fig. 14. Confusion Matrix Model 6

D. *Accuracy vs time taken*

On comparing the accuracy obtained w.r.t time Figure 15, we notice that the shallow forward feed model takes around 40 min to train to provide an accuracy of 52 percent whereas model 6, a two-layer Deep Neural Network model took 3.9 hours to provide an accuracy of 62 percent. Model 2 performed the worst in terms of time consumption where it took 10.5 hours to train and produced an accuracy of 60.61 percent, indicating that more layers don't always mean better accuracy.

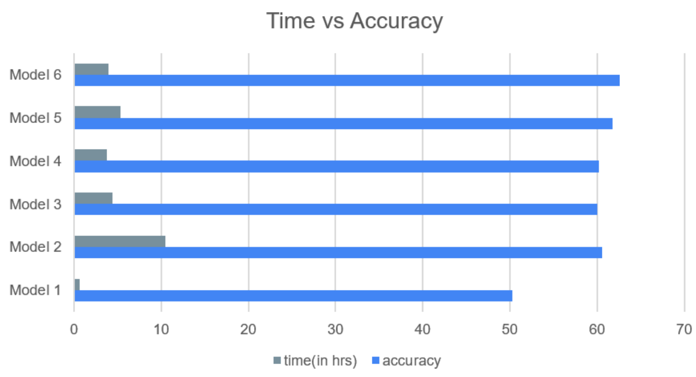

Fig. 15. Time taken for execution vs accuracy

VI. Conclusion

In this experiment in order to find a more accurate way to detect diabetic retinopathy, Deep Neural Network with GA integration is found to provide more accurate results compared to shallow forward feed model. While having the number of layers played an important role where we can observe in model 2 that there were 4 hidden layers and took 10.5 hours to train only to provide 60.61 percent accuracy, whereas model 6 with just 2 hidden layers took almost 4 hours to train to provide 62.9 percent accuracy from Table I. Finding the optimal Neural Network configuration plays a major role in finding the accuracy of the model. We can also conclude that the Genetic algorithm integrated with the Deep Neural Network improves the performance of the model.

References

[1] https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set

[2] https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

[3] https://www.sciencedirect.com/book/9780128219867/nature-inspired-optimization-algorithms

[4] Deb, Kalyanmoy, et al. ”A review on genetic algorithms and their applications.” Frontiers in Robotics and AI 3 (2016): 1-17.

[5] Wang, Jianyong, et al. ”Applications of genetic algorithms in image processing.” Journal of Software Engineering and Applications 7.5 (2014): 337-345.

[6] Yang, Shengxiang, et al. ”A survey of swarm intelligence for dynamic optimization: Algorithms and applications.” Swarm and Evolutionary Computation 9 (2013): 1-24.

[7] Boonmee, Chaivatna, and Chaiyong Ragkhitwetsagul. ”Application of genetic algorithms in financial portfolio optimization: A review.” Journal of Systems and Information Technology 18.3 (2016): 322-342.

[8] Davis, Lawrence. ”Handbook of genetic algorithms.” Springer Science Business Media, 2013.

[9] Khemphila, Aree, et al. ”Application of genetic algorithms in optimization of machine learning models: A review.” International Journal of Computer Applications 168.6 (2017): 9-15.

[10] Li, Xin, and Jingwei Wang. ”Applications of genetic algorithms in scheduling problems.” Journal of Systems Engineering and Electronics 25.2 (2014): 229-240.