Finding optimal parameters and optimization techniques using Genetic Algorithms

Venkatesh Nagasubramanian
6721597
University of Surrey
vn00197@surrey.ac.uk

Shaikh Rezwan Rafid Ahmad
6732715
University of Surrey
sa03267@surrey.ac.uk

Shariq Bashir Shaikh
6748153
University of Surrey
ss03958@surrey.ac.uk

***Abstract* - Deep Learning models and convolution networks are one of the most popular and successful method to achieve image classification. One of the key problems is that the parameters are set based on trial-and-error basis and not by using computational methods. We present a way to select the best parameters and the best optimizers using a Genetic Algorithm to get better accuracy from our model.**

Keywords – Image classification, Neural network, Convolutional layer, Genetic Algorithms, Gradient descent

I. Introduction and Literature review

It is easy for a human to look at an image and classify it accordingly. The human brain is heavily trained to identify, recognize and classify images from an early age. However, computers require to be built a neural network to be trained to classify the contents of an image. Convolutional Neural Networks have been extremely useful to solve image classification problems.

The purpose of this paper is to create a genetic algorithm to tune the parameters and get the best activation function and optimizers while keeping the architecture constant.

The use of CNN (Convolutional Neural Network) is immensely popular for solving the image classification problems. Demirci et. al. [1] have proposed a CNN-based architecture for image classification of the CIFAR 10 dataset with around 86% accuracy, while utilizing only 2 gigabytes of memory. This architecture has 4 Convolution layers which is succeeded by 2 fully connected (dense) layers. Doon et. al. [2] proposed a ConvNet model where they used 6 convolutional layers with a pooling layer for each. Various regularization techniques and dropout layers is added to each layer to reduce overfitting of the model. The model presented a remarkably high validation accuracy of 87.57% while the training accuracy was 90%.

Apart from using the convolutional neural networks for this problem, Genetic algorithms have been used by Aszemi, et. al. [3] to solve this problem. The architecture comprises of a series of neural networks with pooling and the dropout layers as the ones in the previous works, but they have used a genetic algorithm to tune and optimize the hyper parameters that determines the network structure and the network trained. On CPU, the training ran for 5 days only to reach an accuracy of 60.85%. On GPU, the training ran for around 3 days and the accuracy was 71.17%.

Very deep neural network was used by S. Liu, et al. [4] who has modified the VGG-16 neural network to fit a small dataset like ours. The model inferred that the usage of batch normalisation and a strong dropout setting as might increase accuracy, but not the state-of-the-art standard. However, they propose future research is needed to see how deep models can be used for small datasets.

Stochastic gradient descent is one of the most used methods for training deep learning models. The work done by Hsueh et. al. [5] proposes a new method to schedule the learning rate of the gradient. The paper presents an exceptionally low error rate when this method is used for optimizing the learning rate. This is a very good research paper to learn more about adaptive learning rate methods.

Diederik P. Kingma and Jimmy Lei Ba [11] introduced a new method for Stochastic Optimization called Adam optimizer. The Adam optimizer uses a combination of RMSProp and SGD optimizers with momentum. This has been an immensely popular choice for optimizer since it was presented, as per the blog at [12]

While all these models boast of a very high validation and training accuracy, there is no clear justification of why a specific activation function is used for their problem. Most of the papers usually are research for a new optimizer like [11] or explain a new activation function like the work done by [2].

In this paper, we create a genetic algorithm to crossover and select the best combination of the parameters (the activation function) and the optimizer. The main difference is that our algorithm does not tamper with the architecture of the neural network as in the work done at [3].

II. Methodology

A. Dataset

The CIFAR-10 dataset contains 60,000 (32x32) coloured images which are split into 50,000 training and 10,000 testing images. [6] There are 10 classes – airplane, automobile, bird, cat, deer, dog, frog, horse, sheep, and truck. In x_train and x_test, they have values ranging between 0 and 255 across 3 dimensions as they are in RGB, whereas y_train and y_test are values from 0 to 9 which indicates the classes.

Table 1 shows the dimensions of the dataset (training and testing or validation).

Datasets	Dimensions
Training Data (x_train)	(50000, 32, 32, 3)
Training Labels (y_train)	(50000, 1)
Testing Data (x_test)	(10000, 32, 32, 3)
Testing Labels (y_test)	(10000, 1)

Table 1 – Dimension of training and testing sets

B. Data Pre-processing

Normalization

As the values range from 0 – 255 per image in the training(x_train) and testing(x_test) set, this will lead to slower convergence leading to a deficient performance by the neural network.

One Hot Encoding

A method to quantify categorical data by producing vectors with lengths equal to the number of categories. [7] Even though values in our labelled sets (y_train, y_test) are integers (0 – 9) however, they are mapped to classes meaning we can use one-hot encoding.

For instance, if the label for class 'cat' is 3, the binary matrix representation would be [0,0,0,1,0,0,0,0,0,0].

C. Architecture

We have chosen Convolutional Neural Network as our architecture for the three models. It consists mainly of convolutional (CONV2D) and dense layers. The following figure Fig – 1 and Table – 2 describe the CNN architecture with parameters in detail.

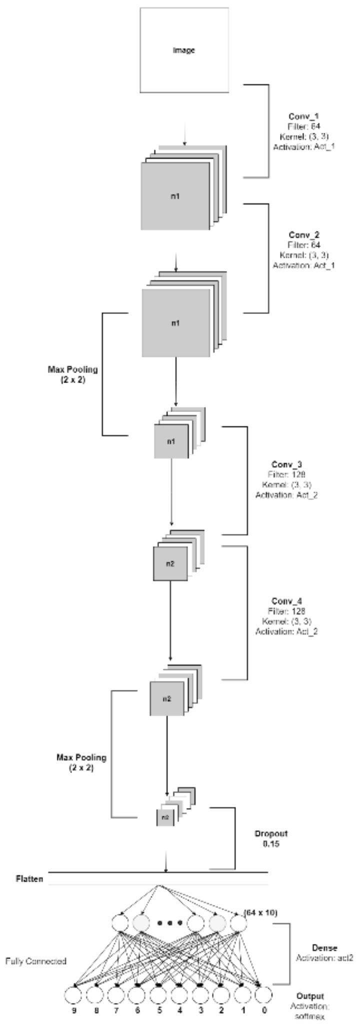

Fig 1 – Architecture of the Convolutional Neural Network

Layer	**Parameter**
Conv_1	Filter: 64, Kernel: (3,3), Activation: act_1
Conv_2	Filter: 64, Kernel: (3,3), Activation: act_1
Max Pooling	(2, 2)
Conv_3	Filter: 128, Kernel: (3, 3), Activation: act_2
Conv_4	Filter: 128, Kernel: (3, 3), Activation: act_2
Max Pooling	(2, 2)
Dropout	Rate: 0.15
Flatten	Converting to 1D
Dense	Neurons: 64, Activation: act_2
Dense (output)	Neurons: 10, Activation: softmax

Table 2 – Parameters of the CNN architecture

The following loss and activation functions are used throughout the three models extensively -

Activation Functions

ReLu: Rectified Linear Unit or ReLu is a linear function that gives the output the same as the input if it is positive or zero if it is negative. This activation layer is used extensively in each hidden layer. [8]

Softmax: The softmax activation function is very commonly used in categorical classifications where it calculates the relative probabilities of each class. [9]. For all the models, this was used in our last layer to classify classes.

Loss Functions

The loss function used in the algorithm is Categorical Cross Entropy. Also defined as softmax loss, it is used to train the neural network to output a probability over the 10 classes [10]. This loss function can be represented mathematically as:

$$f(x) = -\sum_{i=1}^{n} t_i \log(s_i)$$

Where t_i is the ground truth and s_i is the score for each class. n is the number of classes, in our case, it is 10.

III. Models & Results

We have implemented 3 different optimizers as per our CNN architecture in Fig 2. The optimizers are Stochastic Gradient Descent (SGD), Adam and our proposed model Genetic Algorithm (GA) which tries to find the most optimal parameters, in this case activation functions and optimizers.

The epoch size is set to 30 and batch size to 32 and is the same across all the models.

A. Model I - Stochastic Gradient Descent

In our first optimizer, we are using the Stochastic Gradient Descent (SGD) as per the architecture of the neural network in Fig 1. Gradient descent is one of the most used common optimizers in neural network. To search the local minima, the algorithm iteratively finds the local optimum (minima or maxima) for a given function. The $\nabla f(a_n)$ is the loss function and α the learning rate.

$$a_{n+1} = a_n - \alpha \nabla f(a_n)$$

The reason we chose Stochastic Gradient Descent (SGD) over vanilla Gradient Descent (GD) is SGD picks one data point in a random fashion from the entire data set at each iteration which enormously reduces the computations [8].

For act_1 and act_2 the activation function 'ReLu' was used.

The hyper parameters of the Stochastic Gradient Descent (SGD) were configured as follows –

Momentum: Accelerates in the relevant direction, dampening oscillations of noisy gradients. [9]

Nesterov: Calculates the decaying moving mean of the gradients of projected positions in the search space rather than original positions. [10]

The equation to compute the updated weights when momentum is greater than 0 and Nesterov = True is the following [13]

$$velocity = momentum * velocity - learning_{rate} * g$$
$$w = w + momentum * velocity - learning_{rate} * g$$

RESULTS

The model was run three times and the mean training accuracy/loss and validation accuracy/loss is calculated. The execution time to perform this operation is approximately 150 seconds.

The training accuracy is 95.75% meaning the model is not underfitted. However, validation accuracy is 71.66% which is due to overfitting and the validation loss is extremely high 152.6%.

Iteration	Train Acc	Val Acc	Train Loss	Val Loss
1	0.9573	0.7213	0.1245	1.5522
2	0.9585	0.7153	0.1189	1.4805
3	0.9567	0.7131	0.1264	1.5454
Average	0.9575	0.7166	0.1233	1.5260

Table 3 – Metrics of Model I – Stochastic Gradient Descent

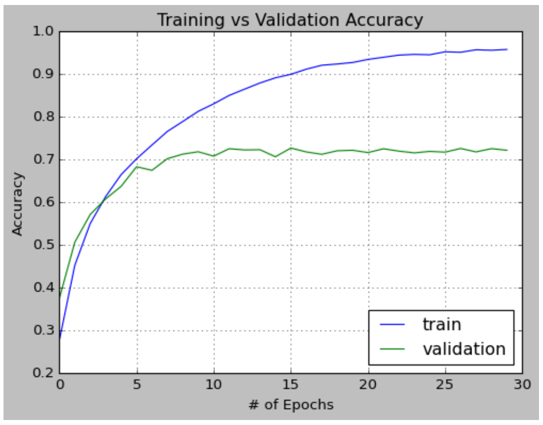

Fig 2.a – Training vs Validation Accuracy of Model I

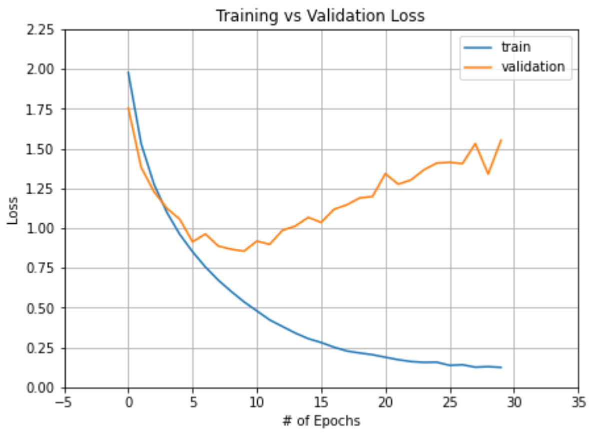

Fig 2.b – Training vs Validation Loss of Model I

B. *Model II – ADAM*

As our second optimizer, we are using the Adam with convolutional layers as per the architecture of the neural network in Fig 1.

Adam (Adaptive Moment Estimation) optimizer is an algorithm for gradient descent optimization technique, combining the best of both momentum and RMSProp, evolved by combining the smoothening ability of momentum and efficient learning rate ability of RMSProp.

ADAM has been the most popular choice of optimization algorithm for image classification in recent years as it converges faster than SGD and with appropriate parameters and hyperparameter tuning, it can generalize better than Stochastic Gradient Descent (SGD). It can be represented mathematically as follows: [15]

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t.$$

RESULTS

A validation accuracy of 72.7% and training accuracy – 92.95% was attained in our 2nd model. It performed similar to our previous model i.e Model I – Stochastic Gradient Descent. Table 4 below shows the training and validation accuracy, as well as the loss per iteration. The execution time per iteration was a bit higher than Model I, approximately 200 seconds.

# Itera-tion	Train Acc	Val Acc	Train Loss	Val Loss
1	0.9232	0.7235	0.2171	1.2297
2	0.9328	0.7259	0.1924	1.2501
3	0.9326	0.7338	0.1946	1.2686
Average	0.9295	0.7277	0.2014	1.2494

Table 4 – Loss/Accuracy metrics per iteration of Model II.

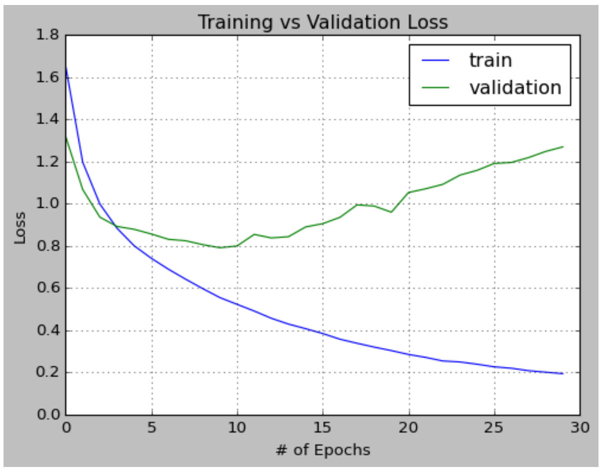

Fig 3.a – Training vs Validation Loss of Model II.

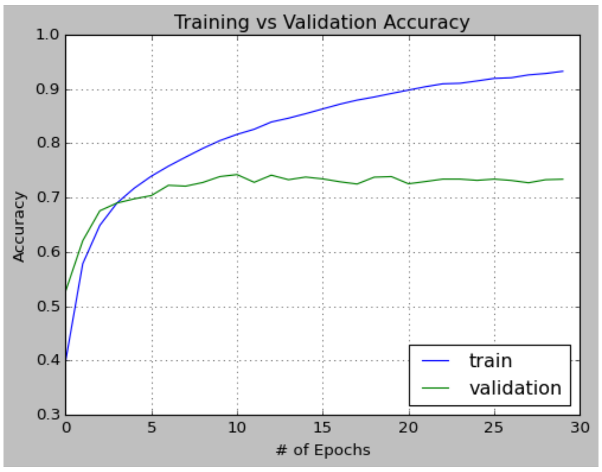

Fig 3.b – Training vs Validation Loss of Model II.

C. *Model III – Genetic Algorithm to find optimal optimizers and parameters*

As our proposed model to find optimal activation functions and optimizers we will be using the genetic algorithm approach.

The Genetic Algorithm (GA) tries to mimic biological evolution. It uses the natural selection process to optimize the problem over the coming generations. It repeatedly modifies each of the candidates in the population to obtain a new generation and selects the best solution to be added as parents for the next generation. We have used the steady-state replacement technique, which carries the 2 offspring and replaces it with the 2 worst individuals from the population (ranked by fitness) of the previous generations to carry on as the population for the next generation.

In our demonstration, we are using a population of 6 neural networks which will run for 5 generations.

Step I – Population and Training

We created a population of 6 neural networks – M1, M2, M3, M4, M5 and M6. Each model is configured with different optimizers – Adam, Adagrad, RMSProp, Adadelta, Adam, SGD and RMSProp. Similarly, 4 different activation functions were set as well – ReLu, SeLu, Elu and lastly sigmoid. The initialization of these of the neural network is configured manually for the first generation only. Table 5 – shows the optimizers, activation functions that were set for each of the 6 models, that were pre-set for Generation 1.

Model	**Activation 1 (act_1)**	**Activation 2 (act_2)**	**Optimizer**
M1	relu	elu	RMSProp
M2	relu	relu	Adam
M3	relu	selu	Adagrad
M4	selu	elu	SGD
M5	relu	sigmoid	Adamax
M6	sigmoid	selu	Adadelta

Table 5 – Optimizers and activation used to generate population in Model III.

The model is then trained as per the CNN architecture in Fig 1.

Step II – Evaluating Fitness

The fitness value is then calculated for every model that is trained, as per the architecture (Fig 2) above. This is the validation accuracy where the testing set (x_test and y_test) is used. The fitness score is then recorded, which will be later used in the next step of the process.

Step III – Selection

Once, all the models are trained and the value of the fitness is calculated, two parents out of the population will be selected. This is like natural selection and is done through a roulette wheel process, where the probability of each candidate will be the fitness proportionate itself. Therefore, the higher the fitness value the higher the possibility of being selected. The pseudocode below briefly describes the roulette selection process.

<u>Pseudocode for Selection</u>

```
Step 1: Get the list of fitness values as fitness list.
Step 2: Initialize fitness percentage list.
Step 3: Initialize the wheel list.
Step 4: for every f in fitness list do
            Append f/sum(fitness list)*100
            End loop
Step 5: while i < length of percentage list do
            Append i to wheel list percentage(i) times
            End loop
Step 6: make a random choice of 2 parents in the wheel
Step 7: return parents' index
```

Step V – Cross-over

In genetic algorithms and evolutionary computation, the crossover is used for combination of the genetic information of both the parents to create new offspring based on a distribution of probability. [14] In the case of neural networks, instead of genetic information, the activation functions and optimizers are changed within the 2 selected models.

<u>Pseudocode Cross-over</u>

```
Step 1: Get the parents p1, p2 from selection
Step 2: Initialize children c1, c2
Step 3: Activation 1 of c1 = chosen randomly from
          (Activation 1 of p1, Activation 1 of p2)
```

Step 4: Repeat above step for c2
Step 5: Repeat Step 3 for Activation 2 for c1 and c2
Step 6: Choose optimization function for c1 from optimization function of p1 and p2.
Step 7: Repeat same for c2
Step 8: return parameters for c1 and c2.

Step VI – Removing networks with low fitness values

After we generate 2 models (child) by crossover in Step V, we add them to our existing population. This will result in a population of size = 8, however, as mentioned we used the steady-state replacement technique, therefore the two models that have the lowest fitness score will be removed for the next generation. The population size before the initiation of every generation becomes 6 again, and the process continues until and unless the threshold accuracy is met or the desired generation is reached.

The following Fig 4 describes the flowchart of the process.

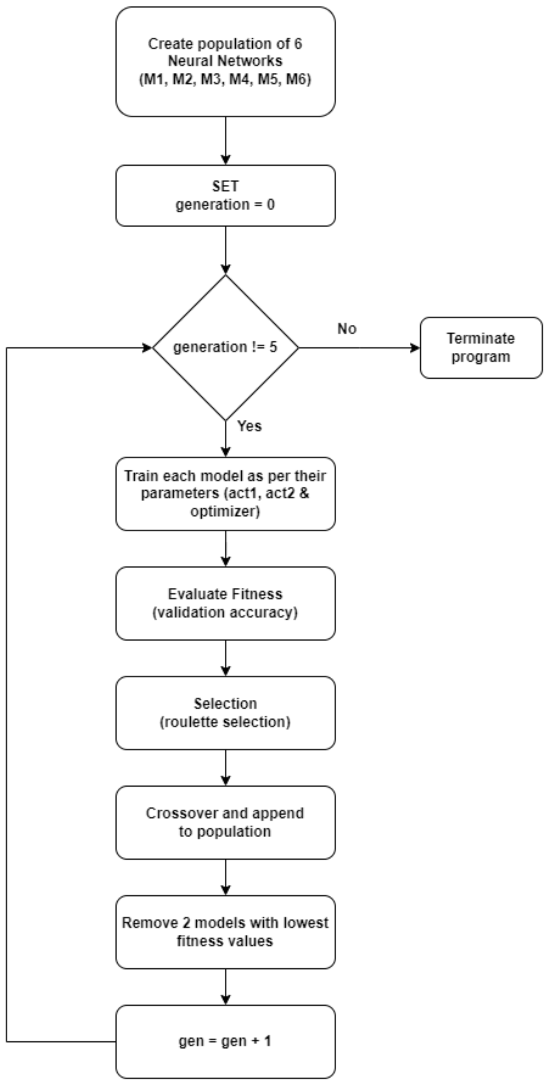
Create population of 6 Neural Networks (M1, M2, M3, M4, M5, M6)
SET generation = 0
generation != 5
No
Terminate program
Yes
Train each model as per their parameters (act1, act2 & optimizer)
Evaluate Fitness (validation accuracy)
Selection (roulette selection)
Crossover and append to population
Remove 2 models with lowest fitness values
gen = gen + 1

Fig 4 – Flowchart of Model III

RESULTS

After running it for 5 generations, the highest validation accuracy we got was for the Adamax optimizer with ReLu and Sigmoid activation functions.

The execution time per generation was 18 minutes on average, totalling approximately 1.5 hours.

The following Table 5 shows the breakdown of the fitness value per model for each generation.

Models	Act_1	Act_2	Optimizer	Fitness
M1	relu	elu	RMSProp	0.746
M2	relu	relu	adam	0.738
M3	relu	selu	adagrad	0.487
M4	selu	elu	SGD	0.666
M5	relu	sigmoid	adamax	0.775
M6	sigmoid	selu	adadelta	0.273
Winner	relu	sigmoid	adamax	0.775

Generation 1

Models	Act_1	Act_2	Optimizer	Fitness
M1	relu	elu	RMSProp	0.716
M2	relu	relu	adam	0.73
M3	selu	elu	SGD	0.667
M4	relu	sigmoid	adamax	0.765
M5	relu	selu	SGD	0.657
M6	relu	relu	adamax	0.766
Winner	relu	relu	adamax	0.766

Generation 2

Models	Act_1	Act_2	Optimizer	Fitness
M1	relu	elu	RMSProp	0.729
M2	relu	relu	adam	0.726
M3	relu	sigmoid	adamax	0.77
M4	relu	relu	adamax	0.77
M5	selu	selu	SGD	0.704
M6	selu	selu	SGD	0.68
Winner	relu	relu	adamax	0.77

Generation 3

Models	Act_1	Act_2	Optimizer	Fitness
M1	relu	elu	RMSProp	0.739
M2	relu	relu	adam	0.723
M3	relu	sigmoid	adamax	0.773
M4	relu	relu	adamax	0.758
M5	relu	relu	RMSProp	0.696
M6	Relu	Elu	RMSProp	0.698
Winner	relu	sigmoid	adamax	0.773

Models	Act_1	Act_2	Optimizer	Fitness
M1	relu	relu	adam	0.7
M2	relu	sigmoid	adamax	0.771
M3	relu	relu	adamax	0.768
M4	relu	elu	RMSProp	0.733
M5	relu	sigmoid	adamax	0.77
M6	relu	sigmoid	adamax	0.764
Winner	relu	sigmoid	adamax	0.77

Generation 4
Generation 5

Table 5 – Metrics for M1 – M6 for Generation 1 to 5.

By using the Adamax optimizer with the ReLu & Sigmoid activation function pair we were able to identify the best possible combinations for our architecture.

IV. CONCLUSION

In this paper, we have implemented three algorithms, Stochastic Gradient Descent, a Genetic Algorithm and a 2D Convolutional Neural network with the Adam optimizer Our main goal was to implement a Genetic Algorithm to maximize the accuracy by automatically tuning its parameters.

The other models that we had implemented had similar validation accuracies. The validation accuracy for the SGD model and the Adam optimizer was 71.66% and 72.7% respectively. The activation function for this was chosen using a trial-and-error process and had no justification for this choice. However, we implemented a Genetic Algorithm which gave us a better accuracy of over 77%. This is because the GA has chosen, with a series of crossovers and selection algorithms to make the accuracy better over five generations.

There is scope for more research on this project. The GA does help in getting a higher accuracy for our data but does not account for the overfitting of the data. With more research on this topic, we can account for the fit of the data and convert it to a bi objective optimization problem where we try to maximize the accuracy while minimize the difference between the training and the validation accuracy.

V. REFERENCES

[1] R. C. Çalik and M. F. Demirci, "Cifar-10 Image Classification with Convolutional Neural Networks for Embedded Systems," 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA), 2018, pp. 1-2, doi: 10.1109/AICCSA.2018.8612873.

[2] R. Doon, T. Kumar Rawat and S. Gautam, "Cifar-10 Classification using Deep Convolutional Neural Network," 2018 IEEE Punecon, 2018, pp. 1-5, doi: 10.1109/PUNECON.2018.8745428.

[3] Nurshazlyn Mohd Aszemi and P.D.D Dominic, “Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms” International Journal of Advanced Computer Science and Applications (IJACSA), 10(6), 2019.

[4] S. Liu and W. Deng, "Very deep convolutional neural network-based image classification using small training sample size," 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), 2015, pp. 730-734, doi: 10.1109/ACPR.2015.7486599.

[5] B. Hsueh, W. Li and I. Wu, "Stochastic Gradient Descent with Hyperbolic-Tangent Decay on Classification," in 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), Waikoloa Village, HI, USA, 2019 pp. 435-442. doi: 10.1109/WACV.2019.00052

[6] California Institute for Advanced Research, Alex Krixhevsky, < https://www.cs.toronto.edu/~kriz/cifar.html >

[7] One Hot Encoding, DeepAI, < https://deepai.org/machine-learning-glossary-and-terms/one-hot-encoding >

[8] Stochastic Gradient Descent — Clearly Explained !!, Towards Data Science, Aishwarya V Srinivasan, < https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31 >

[9] SGD, Keras.io, < https://keras.io/api/optimizers/sgd/ >

[10] Gradient Descent With Momentum from Scratch, Machine Learning Mastery, Jason Brownlee, < https://machinelearningmastery.com/gradient-descent-with-momentum-from-scratch/ >

[11] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[12] Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, Jason Brownlee, Machine Learning Mastery, < https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ >

[13] TensorFlow documentation, < https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/experimental/SGD >

[14] Crossover (genetic algorithm), Wikipedia, < https://en.wikipedia.org/wiki/Crossover_(genetic_algorithm) >

[15] A 2021 Guide to improving CNNs-Optimizers: Adam vs SGD, medium, Seiun Park, < https://medium.com/geekculture/a-2021-guide-to-improving-cnns-optimizers-adam-vs-sgd-495848ac6008>

VI. APPENDIX

The tasks were divided in such a way that each member had somewhat equal contribution towards the coursework and allocations were done keeping that in mind.
All the members were present in each of the discussion and meetings and contributed to the best of their knowledge and skills.
For the literature review in section 1, each member studied two academic literatures and the corresponding literature review was written in the report.
As for section 4 where the implementation of algorithm takes place, each one of us were involved in the understanding of the data and pre-processing steps, and as for the three algorithms and its implementation, they were taken by each of the member as follows: -

1. Shariq Bashir Shaikh was responsible for Stochastic Gradient Descent
2. Venkatesh Nagasubramanian was responsible for Adam optimizer algorithm
3. Shaikh Rezwan Rafid Ahmed was responsible for Genetic algorithm

The codes were written and implemented in Google Colab notebook and Draw.io was used to create the flowchart diagrams.

The rest of the report and its findings were made in collaboration and with consensus of all the members. The code was made in the python language using the Tensorflow libraries. The IDE used was google colab pro.