The influence of genetic algorithms on hyperparameter optimization for neural networks

Catrina Mirela, Catrina Sorana, Ghiurțu Andrei

Table of contents

Abstract **1**

Problem statement **1**

Background and related work **2**
- Theoretical framework 2
  - Convolutional Neural networks 2
  - Genetic algorithms 2
- Related work 3

Discussions and analysis **5**
- Choosing the network 5
- Network description 6
- Genetic Algorithm overview 6
  - The genetic code 6
  - The selection method 7
  - The crossover method 9
  - The fitness function 10
  - Mutation 10
  - Putting it all together: Building the next generation 11
- Proposed approaches 12
  - The first test 12
  - The second test 14
  - The third test 16
  - The fourth test 18

Conclusions and future work **19**

References **21**

Appendices **23**
- Appendix A: Possible NNet analysis - detailed layers 23

1. Abstract

Machine learning covers a broad range of neural networks: CNNs, RNNs, Feed Forward NNs etc. When it comes to image classification, the Convolutional Neural Network is by far the most used one. The problem that comes with choosing this architecture is the large number of parameters that the developers must define. Choosing the best numbers to use for such parameters is a time-consuming task since the only way of doing so is by running numerous tests and comparing the outputs; even so, one can not be sure that the chosen parameters yield the best possible answer.

Our study decided on the approach of using genetic algorithms to change such parameters. By encoding the parameters in an efficient way, we allow the genetic algorithms to evolve and yield the best result. The proposed algorithm is examined on a CNN developed for flower recognition that trains on flower photos provided by Google for their neural network consisting of 3700 images. The experimental results demonstrate the advantages of the proposed approach; a rise in the accuracy is visible. We predict that using the same approach in more complicated CNNs that train on computers with better resources should yield similar results.

2. Problem statement

We studied the problem of generating the hyperparameters for a neural network, specifically a Convolutional Neural Network. The current broadly-used approach is manual selection. This method is not the most efficient one since there is a constant need of human resources to adjust the architecture.

The aim of this paper is to successfully implement a Genetic Algorithm that deduces an optimal architecture for a CNN of choice. The chosen GA structure should be able to discover both the number of layers necessary and the type of layers to use without any manual intervention. To achieve our goal, we will explore different selection methods, gene encoding parameters and generation settings in order to give out GA the best chances to yield the expected result.

Our approach, alongside other related works, should demonstrate the utility of Genetic Algorithms in choosing hyperparameters for a neural network and improve the time necessary for implementing an optimal architecture for any CNN.

3. Background and related work

3.1. Theoretical framework

3.1.1. Convolutional Neural networks

Regarding deep neural networks, Convolutional Neural Networks have shown great success in a variety of problems consisting of image and video recognition, speech recognition and natural language processing. This neural network's name indicates the mathematical operation that underlies the learning process, called convolution.

CNNs are built, essentially, using four operations: the convolution, activation function, pooling or sub sampling, and classification.

The convolution operation is the one that extracts the features of the image that is fed into the NN. It uses filters, also called 'kernels' or 'feature detectors', that slide over the image and receive a matrix called 'feature map'. The values of these filters are changed during the training process. Usually having more filters provided by more layers results in a better outcome as they are able to identify more features.

The activation function rectifies the feature map. The activation function is chosen depending on the input provided. The most known are ReLU and the sigmoid function [4]. ReLU (Rectified Linear Unit) is the most used activation function since 2017 in deep neural networks since it has shown great success in a great variety of neural networks. This activation function does not change the positive values in the feature map, replacing just the negative ones with 0.

The pooling is used to retain only the most important information from the given feature map. Usually average or max pooling are the possibilities for this operation. Max pooling works by applying a filter that chooses the maximum value of non-overlapping subregions in the matrix provided [6]. It is an important step since it reduces the dimensions and allows only important information to matter for the outcome.

The classification is the one that gives the output. The previous step provides the image features; these are analyzed by the fully connected layers that choose the category that best fits the image.

3.1.2. Genetic algorithms

Genetic Algorithms are heuristic solution-search or optimization methods. Intuitively, they work in a similar way to how natural evolution works. The fundamentals of Genetic Algorithms were developed based on the same ideas as the Darwinian principle of natural selection.

GAs work on multiple generations. Each generation is composed of a number of individuals (each individual represents a possible solution for the problem set out to solve). For each individual a fitness function is evaluated. This function tells the algorithm how good a certain individual performed on the given task. After all the individuals from a generation are evaluated, a new generation is formed, usually based on the crossover of the individuals that had a good performance from the previous generations.

The GA begins by generating a random initial population of individuals. Each individual is represented through a gene (or chromosome) that encodes the possible solution for the problem. After this, as mentioned earlier, through a process of selection and recombination new generations are formed. The idea is that the selection algorithm and the crossover method will ensure that each new generation will perform better than the previous one.

Therefore, the steps of a genetic algorithm can be divided as follows:

1. Randomly generating the first generation
2. For each generation that follows:
    a. Calculate how good each individual performs on the given task (calculating the fitness function)
    b. Through the selection method, certain parents are chosen to produce offspring
    c. Using the crossover method, the offspring are produced and added to the new population
    d. The current population is discarded, and the new population gets considered

Moreover, we can also see that each GA has certain "standard" components:

1. The encoding of the hyperparameters in the genetic code (the code for each individual)
2. The selection method
3. The crossover method
4. The fitness function
5. The mutation rate

Also, another important factor is deciding the number of individuals per generation and the number of generations.

3.2. Related work

The problem of the time-consuming manual hyper-parameter optimization has been observed since the beginning of the field. Because of this, multiple studies have been actively carried out to propose different solutions to this problem. The use of Genetic Algorithms has also been studied on multiple occasions [1, 2, 3, 4, 7, 9, 10]. The difference between these studies consists in different gene encoding, different selection and crossover methods, different fitness functions etc, each of them trying to reach the best hyper-parameter optimization for the studied network.

One big advantage to Genetic Algorithms is that they are very versatile and can be used in hyperparameter optimization for a lot of problems, and therefore, for different Neural Networks. For example, Yanan Sun et al. used GA for optimizing the selection of parameters for a CNN trained on the MNIST datasets [1], while Edson D. Carvalho and collaborators used GA for the same purpose but on a CNN trained to classify CT scans for Non-Covid and Covid infected patients [2].

Moreover, even if the GA has certain “fixed” components, these can also be improved and adapted for the problem at hand. For example, while considering the fitness function, Ji-Hoon Han et al chose to consider the accuracy as well as the verification time in order to give the GA an incentive to choose networks that have a smaller verification time [10]. Furthermore, GA can be used in choosing different sets of parameters, depending on the importance of that said parameter in the given problem. For example, one can choose to use GA in order to find the optimal number and order of layers, as done by Yanan Sun and collaborators [1] or/and the hyper-parameters used in the training session (number of epochs, batch size, etc) [3].

Considering these advantages, we chose to use Genetic Algorithms in order to explore how they influence the process of choosing hyperparameters for a given Neural Network.

4. Discussions and analysis

4.1. Choosing the network

Bearing in mind that training a neural network has high hardware demands and that the computers we used for this study are not that efficient, we had to choose a neural network that has as little training time as possible so that we can create more individuals (set of parameters) to use during the training of neural network using genetic algorithms. Two other factors were the base accuracy and the number of hyperparameters that could be changed in order to get better results. We had to choose from 5 very well-known and documented neural networks:

NN purpose	**DB size**	**NN architecture**
Flower Classification [8]	3k-4k img.	3*(conv2d + maxpooling) + 2*dense
Movie Review [5]	44k voc. size 1300 max len	$\left.\begin{matrix} conv1d + maxpooling \\ conv1d + maxpooling \\ conv1d + maxpooling \end{matrix}\right\} concatenate + 2 * dense$
Cifar DataSet	25k/60k	2*conv2d(32) + maxpooling + 2*dense
		2*(2*conv2d(32) + maxpooling) + 2*dense
		2*(2*conv2d(32/64) + maxpooling) + 2*dense
		conv2d(32) + conv2d(64) + maxpooling + 2*dense
Digit classification		conv2d + maxpooling
Handwritten Text Recognition	50k/90k	5 * conv2D + BidirectionalRecurrentLayer(LSTM) + CTC
see detailed layers in Appendix A		

Analyzing this table we came to the conclusion that the only two that we could use to try genetic algorithms to improve the training accuracy on the Movie Review and the Flower Classification because of their training time for each epoch and the range for progress in accuracy they have. We chose to go further using the Flower Classification, because the Movie Review architecture is pretty simple and there were not many hyperparameters that could be improved.

4.2. Network description

The architecture chosen by the Tensorflow team for the Flower Classification NN has a sequential model that uses three CNNs, each one followed by a max pooling layer [8]. At the end, the output is flattened and sent to a dense layer. The model compilation uses an Adam optimizer and watches the accuracy metric. Each convolutional layer has a different number of kernels (or filters), starting with the first layer having 16 kernels and the following ones 32 and 64 kernels. Both the CNNs and the dense layer use the ReLU activation function. The output layer has five neurons (flowers), each one knowing the chance that the input image is a type of that flower (neuron), the only possible outputs being: daisy, dandelion, roses, sunflowers and tulips. The model was created using Keras and Tensorflow.

4.3. Genetic Algorithm overview

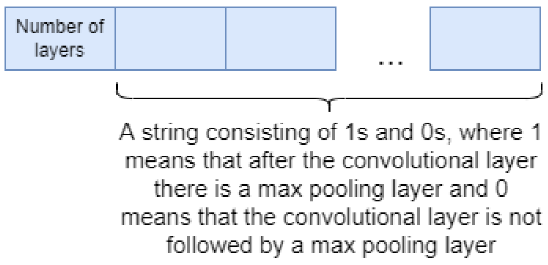
As mentioned in section 4.1.2, in order to implement the Genetic Algorithm for our problem we needed to choose the implementations for each of the five components.

4.3.1. The genetic code

The goal of the project is to see how we can use genetic algorithms for hyperparameter optimization. For the selected architecture, we had multiple options for the information that can be modified through the genetic algorithm:

- modifying the number (and the order) of layers [1]
- modifying the number of kernels for each layer [4]
- what filters are considered when choosing the output of the NN [2]
- batch size [3]
- number of epochs [3]
- optimizer [3]
- number of neurons/fully connected layer [3]

For the first tests we chose a chromosome that encodes the number of convolutional layers and the types of layers (convolution or max pooling layers). The structure is illustrated in the following image:


Number of layers
...
A string consisting of 1s and 0s, where 1 means that after the convolutional layer there is a max pooling layer and 0 means that the convolutional layer is not followed by a max pooling layer

We chose this structure because it will need a smaller number of generations to reach a better solution than the randomized first. We needed this because the computing resources of our laptops is rather small to run the algorithm with more complex chromosomes and the goal of the project is to see how this approach influences the performance of the Neural Network.

The next step after deciding on the genetic code is creating the individual – a CNN built according to its given chromosome – in order to be able to assess the fitness function by training it on the chosen dataset. Our method is shown in Algorithm 1.

Algorithm 1: Generate model

```
Input:    Gene
Output:   Model
1  initialize model
2  for i ← 1, gene[0]
3      add convolutional layer
4      if gene[i] = 1
5          add max pooling layer
6  add flatten model
7  add dense layer
8  compile model
9  return model
```

4.3.2. The selection method

Genetic algorithms can use different selection methods based on the problem at hand. For our project we restricted the choice between the following types of selection algorithms:

- **Roulette wheel selection** [2, 7, 3]: The roulette wheel selection can be visualized as a pie chart in which each section size is proportional to the individual's fitness. Therefore, the bigger the size, the higher the chance for the individual to get selected as a parent. In consequence the individuals with better results have a higher chance of having their genetic code transmitted to the next generation, while also maintaining diversity in the population by allowing individuals with worse performance to be chosen. Spinning the roulette (by choosing a number) the algorithm returns the selected parents for the new population.
- **Tournament selection** [1]: A tournament was a chivalrous competition or mock fight in the Middle Ages and Renaissance. The tournament selection method uses the same idea. There

are multiple implementations of this algorithm. One option consists in choosing the tournament pairs randomly as parents. Another option can be thought of as a two-phase tournament (firstly, multiple individuals are randomly selected from the population, and then the "tournament" takes place; secondly, the winners are chosen – the "tournament" is won by the best two individuals based on fitness and they are sent to produce offspring for the new population).

- **Using an elitist approach** [4]: As it appears from the name of the selection method, the elitist approach selects the best individuals from a generation and saves them directly into the new population.
- **Rank selection:** While it has the same principle as the roulette wheel selection, the rank selection method uses a ranking function instead of using just the fitness to determine the size of each chart chunk. This function has to balance the sizes to reduce the probability of premature convergence. However, this method is more computationally expensive.

For our experiments, we chose to test the GAs using firstly the tournament selection algorithm and, secondly, a combination between tournament and elitism. We decided on testing different selection methods in order to decide on which one fits our case the best.

The tournament selection method implementation is shown in Algorithm 2. Therefore, our approach is similar to the second option presented above.

Algorithm 2: Tournament selection

```
    Input:    Population
    Output:   Parents
1   parents ← pick 4 random individuals from current population and sort them
2   parents ← sort parents by fitness descending
3   return parents
```

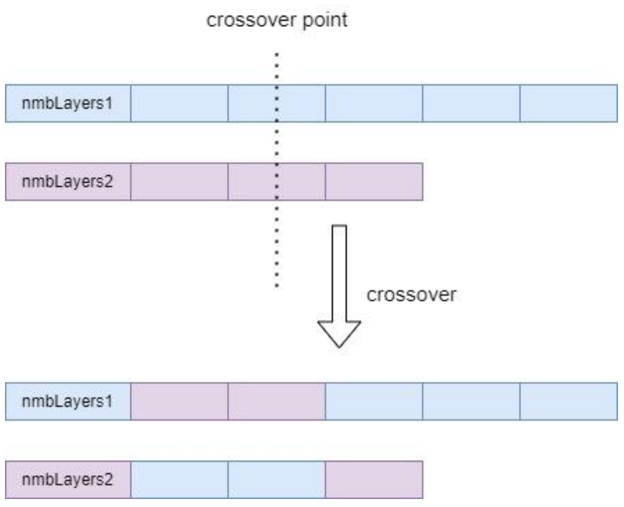
For the last test we chose an approach that also includes elements similar to the elitism selection method. Therefore, before applying the tournament selection we choose a few best individuals that get passed on to the next generation unmodified. This selection method is shown in Algorithm 3. Our idea of selection has the tournament selection principle presented in Algorithm 2, but combined with elitist elements.

```
Algorithm 3: Elitism selection
Input:    Population, k
Output:   The best k individuals that are passed to the next generation unmodified
1  parents ← sort population by fitness descending
2  return first k parents
```

4.3.3. The crossover method

As mentioned in section 3.1.2. (Theoretical framework – Genetic Algorithms), the crossover method consists in the creation of offspring from the two chosen parents. There are multiple methods that perform this action (for example using one, two or more pivot points). The fundamental idea is to combine genetic code from both parents in order to generate new individuals.

We decided on choosing a one-pivot method. This is implemented by randomly generating a crossover point between the first position and the minimum between nmbLayers1 and nmbLayers2 (the number of layers for the two chromosomes chosen to create offspring). The pivot index sections the genetic code of each parent into two parts. We use these components to create the offspring: we keep the second segment of each parent in place and swap the first segments, as illustrated in the image below (the individuals are represented by their genetic code – each rectangle represents a bit of the chromosome: 0 or 1, as explained in previous sections).



Therefore, after randomly generating the crossover point, the implementation is straightforward (Algorithm 4).

Algorithm 4: Crossover

```
Input:  Parents, crossover point
Output: Offspring
1 offspring ← copy size of parent1
2 offspring ← copy elements from 0 to crossover_point from parent2 and crossover_point to end from parent1
3 return offspring
```

In this manner, we will end up with two children for each selected pair of parents, so we need to repeat this process for n/2 steps (where n is the number of individuals in the generation).

4.3.4. The fitness function

For the fitness function we chose the validation accuracy from the training session. Therefore, after the model is created from the given gene, the neural network gets trained on the chosen dataset.

4.3.5. Mutation

An important factor that might reduce the performance of the Genetic Algorithm is the stagnation of the individuals, the problem of the population becoming too uniform, of the individuals becoming too alike. To solve this problem, each time an offspring is created, there is a chance of its gene being mutated, similar to how it happens in nature. By doing this, diversity is brought up into the population, and the convergence of the population is delayed. Mutation is usually applied to the gene, randomizing the chance of a bit to change its value.

Our implementation of the mutation method consists of randomizing a number between 0 and 100 and comparing it to a mutation rate. Having in mind that the mutation can take place in either the size of the NN or in the presence of a max pooling layer, we chose to take different approaches: if the mutation point is on the bit corresponding to the size of the NN, we randomize if we should delete or add a different layer. Otherwise, we flip the bit at the selected point.

Algorithm 5: Mutation

```
Input:  Mutation rate, offspring
Output: New offspring
1 mutation_point ← random number from 0 to size of offspring
2 is_mutated ← random number from 0 to 100
```

```
3  **if** mutation_rate * 100 > is_mutated
4     **if** mutation_point > 0
5        offspring[mutation_point] ← 1 - offspring[mutation_point]
6     **else**
7        rand ← random number from 0 to 1
8        **if** rand = 1
9           offspring ← delete last element from offspring and decrease size of offspring
10       **else**
11          offspring ← add random element (0 or 1) to offspring and increase size of offspring
12 **end**
13 **return** offspring
```

4.3.6. Putting it all together: Building the next generation

Having discussed the components of our Genetic Algorithm, the final step in developing a functional and efficient GA is to create the main function that assesses the current population and updates it. Our approach is shown in the table below, in Algorithm 6.

```
**Algorithm 6:** Update population
**Input:**   Current population and their fitness, number of individuals in the population and
             the mutation rate
**Output:**  The new population after selection and mutation
1  new_population ← initialize new_population using the chosen selection method
2  **for** i ← 0, nmb_ind/2:
3     parent1, parent2 ← select parents using the chosen selection method
4     crossover_point ← pick a random number from 0 to min (size of first parent, size of second
   parent)
5     off1, off2 ← create offspring from (parent1, parent2) using the crossover function
6     apply mutation to off1 and off2
7     add off1, off2 to new_population
8  **end**
9  **return** new_population
```

Now all the important functions have been implemented. All that is left to do is to implement the algorithm that trains the model for each individual and updates the population according to the fitnesses given after training.

Algorithm 7: Training all generations

```
Input:   Number of generations, number of individuals for each generation, number of
         epochs and mutation rate
Output:  Trains all generations, finding the best architecture
1  population ← generate random initial population
2  fitness ← initialized with zeros for all individuals
3  for gen in range of number of generations:
4      for each individual in current population
5          model ← generate model using individual's gene
6          train the model and return model's accuracy
7          fitness[index_individual] ← model's accuracy
8      population ← update generation using Algorithm 6
9  returns individual with the best fitness - the best architecture
```

This is the main function, it is called when a new training session starts with the wanted parameters.

4.4. Proposed approaches

4.4.1. The first test

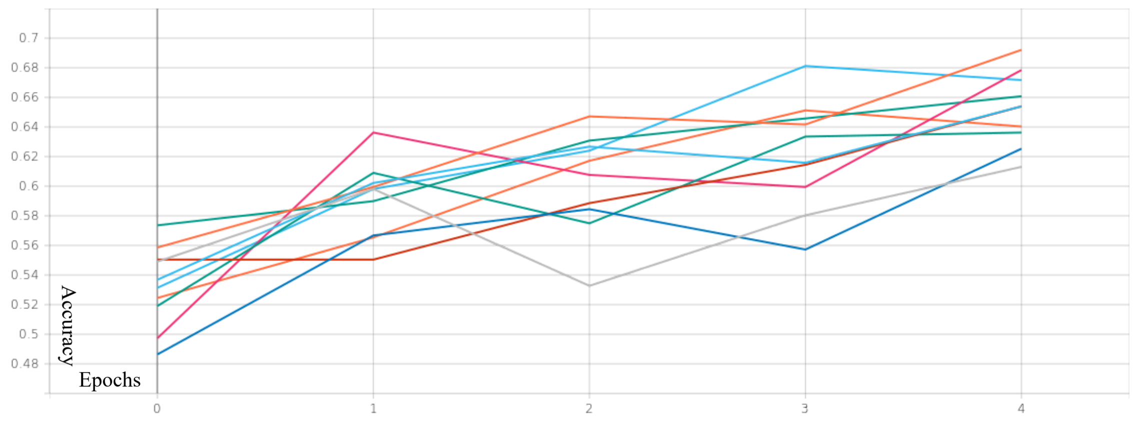
Phase	GA parameters	Values
Initial parameters	Number of generations	5
	Population size	10
	Number of epochs for each individual	5
	Mutation rate	0.20
	Selection method	Tournament
Results	● **The mean accuracy of the population remained the same**	

Proposed changes for better performance	● Larger population size/bigger number of generations ● Different selection method ● Increased mutation rate

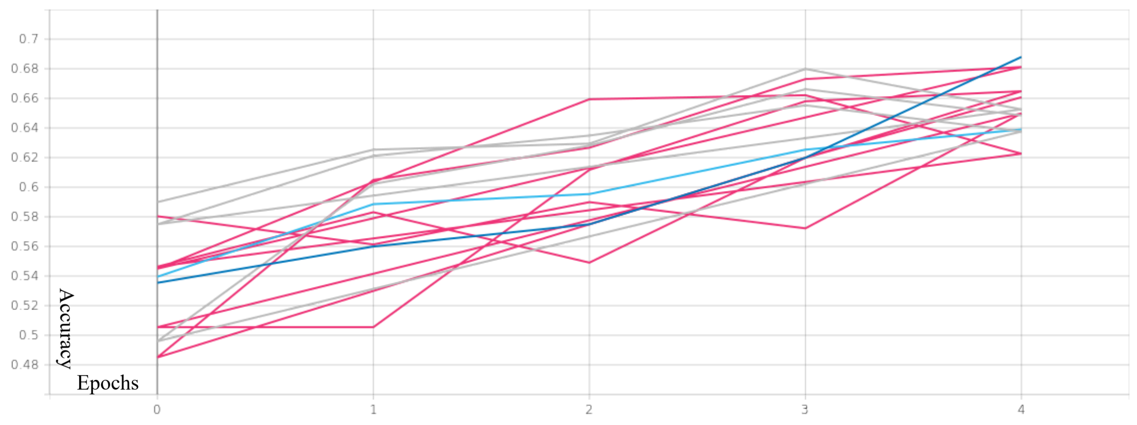
The results of the first test show no improvement in the mean accuracy of the populations for the proposed problem. For this reason, we had to question ourselves if either the initial parameters were wrong, the selection method is not what suits this type of application or both. Our intuition said that the mutation rate should be higher, so that diversity can be brought into the generation. Also, we can improve the selection method.

The biggest change however might come from either choosing a larger sample for the initial population or choosing to let the GA run for a longer period of time (for more generations). For the next test, we chose to change the population size, since we were afraid that letting it run for longer periods of time would not improve the results much since the individuals could become too similar to one another, making it stagnate.

To illustrate this, we generated the following graphs using tensorboard and the logs generated after each individual training. On the X axis we plotted the number of epochs and on the Y axis the validation accuracy for each individual.



Epoch accuracy for the first generation



Epoch accuracy for the last generation

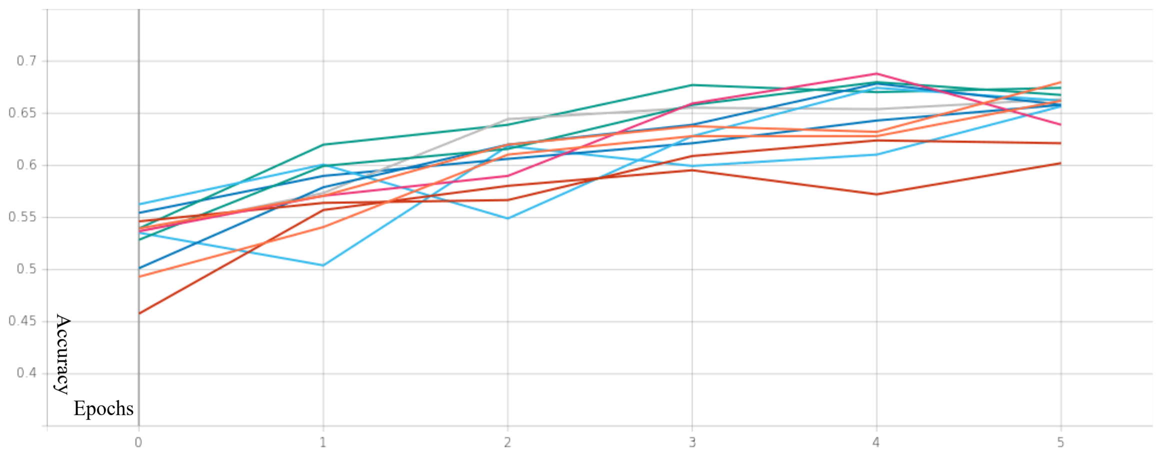
4.4.2. The second test

Phase	GA parameters	Values
Initial parameters	Number of generations	5
	Population size	12
	Number of epochs for each individual	6
	Mutation rate	0.3
	Selection method	Tournament
Results	● **The mean accuracy of the population increased** with each generation that passed, starting from 64% and reaching 67.2% in the last generation, proving that the Genetic Algorithm is efficient in creating better individuals ● By observing each generation pass, we can see that the individuals with the best performance have more max pooling layers (and therefore in each generation there are more individuals with more such layers)	
Proposed changes for better performance	● A bigger number of epochs ● Instead of using the default tournament selection algorithm, we could implement an elitist approach for a fraction of the population, meaning that few of the best will go in the next generation unmodified. We think this will make a change especially because the small number of individuals per generation give the best individual greater importance. ● Considering the selection algorithm, when two individuals have very close accuracies we could prefer to choose the one with a smaller number of parameters	

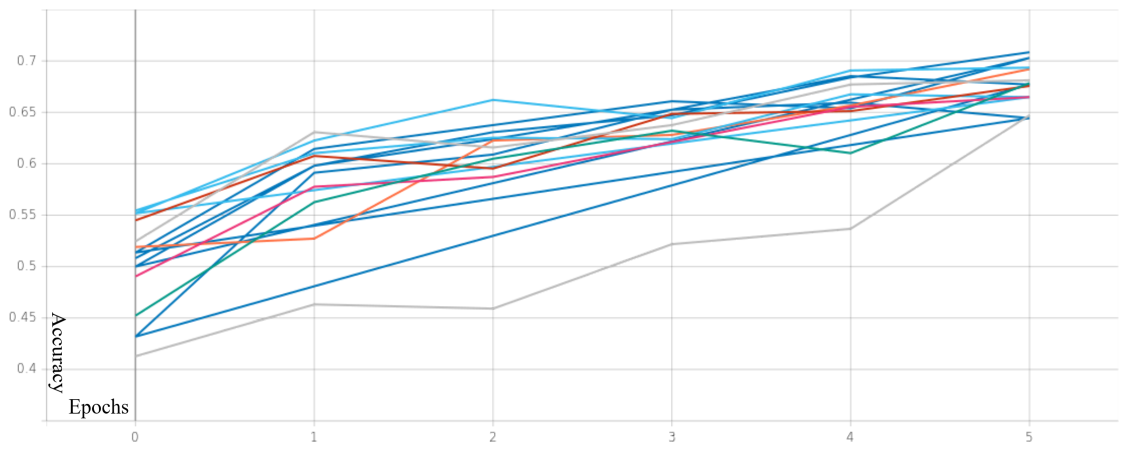
The main purpose of our tests is to see how the use of Genetic Algorithms influences the chosen hyperparameters. Different from the last test, in this experiment a rise in mean accuracy could be observed. We think this is because we chose a larger population size (increasing it by 20%) along with a higher mutation rate. These parameters affected how the GA evolved because the first population was now more diverse and therefore, there was a better chance of finding an individual who performs better.

Due to low computational resources, in the second test, similar to the first one, for each individual we chose 6 epochs to train the neural networks. For this reason, the networks didn't have enough time to reach the best accuracy for their architecture. However, these results can still prove that GA are useful because the increase of the accuracy can be observed even on the small test that we run.

The tensorboard generated the following graphs using the logs related to the second test. The first image shows the epoch accuracy for the individuals from the first generation, while the second one contains the results for the last generation.



Epoch accuracy for the first generation



Epoch accuracy for the last generation

As shown by the two graphs, the mean accuracy increased throughout the generations, demonstrating that the genetic algorithm is efficient at choosing the best architecture for the proposed NN.

Another interesting observation is that if we consider the 10 best individuals out of all individuals of all generations, 60% of them come from the later generations, meaning that the GA actually has a positive impact on the genes of the population.

From these results we can infer that using GA for choosing hyperparameters will yield optimal choices for the parameters encoded. Also, letting the GA run for longer periods of time or/and letting each individual train for more epochs will give a better outcome.

4.4.3. The third test

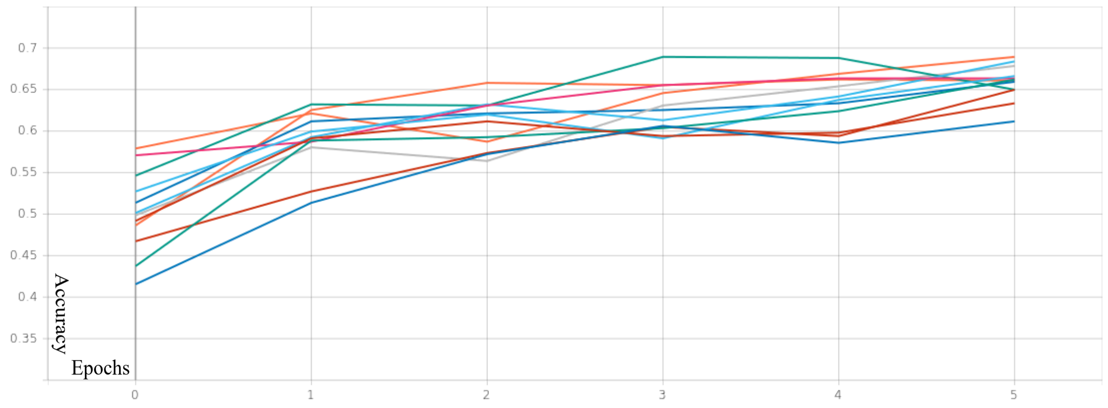
To make sure that the last test was conclusive and did not evolve just by random chance, we chose to rerun it with the same parameters.

Phase	GA parameters	Values
Initial parameters	Number of generations	5
	Population size	12
	Number of epochs for each individual	6
	Mutation rate	0.3
	Selection method	Tournament
Results	● **The mean accuracy of the population also increased,** from 65% to 66.8%	

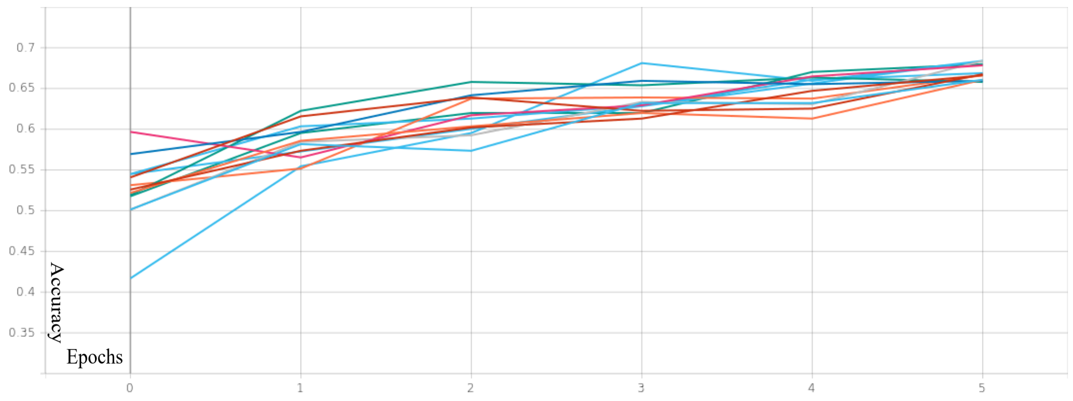
Proposed changes for better performance	● Increase population size to maximize diversity ● If two individuals have almost the same fitness function, we should choose the one with a simpler gene

Comparing the graphs below it can be seen that an increase in the mean accuracy of the generation took place. This test proves that the second test's results were not luck, they are repeatable. By extrapolating, using a more powerful machine to run the Genetic Algorithm, with more epochs and generations, the mean accuracy will increase even more.

An interesting find was that most of the individuals reached almost the same accuracy, which can mean that we should take greater care of the problem of a generation composed of way too similar individuals.



Epoch accuracy for the first generation



Epoch accuracy for the last generation

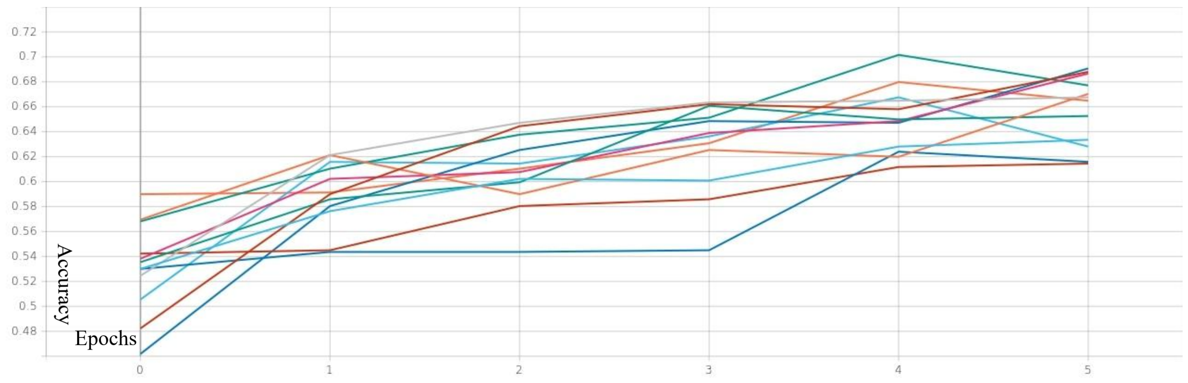
4.4.4. The fourth test

In the fourth test, we chose to implement an elitist-type approach for a fraction of the population, meaning that we will keep few of the best individuals from the previous generation unchanged for the next generation.

Phase	GA parameters	Values
Initial parameters	Number of generations	5
	Population size	12
	Number of epochs for each individual	6
	Mutation rate	0.3
	Selection method	Elitism + Tournament
Results	● **The mean accuracy of the population also increased,** from 65% to 67.4% in the last generation, with the highest mean accuracy reached of 68.2% in the fourth generation	
Proposed changes for better performance	● Increase population size to maximize diversity ● If two individuals have almost the same fitness function, we should choose the one with a simpler gene	

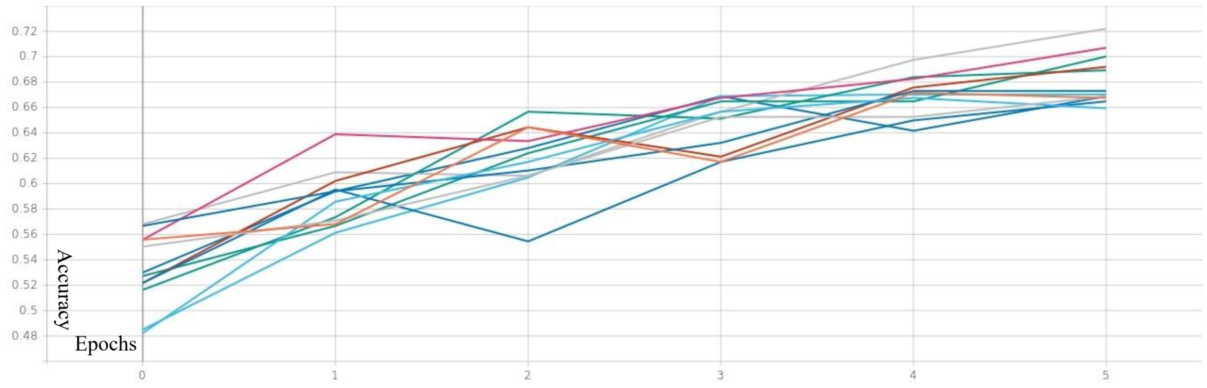
Although a rise in the accuracy over the generations can be observed, we conclude that this is not purely determined by the elitist approach that was added. However, increasing the number of generations, population size and number of epochs might result in a different output, but these will increase the computation power needed in order to get the most out of GA.

Another important result that reveals that an elitist approach may work better for our scenario is related to the highest accuracy reached. During this test we discovered the best individual for our CNN, compared to all individuals in previous other tests.

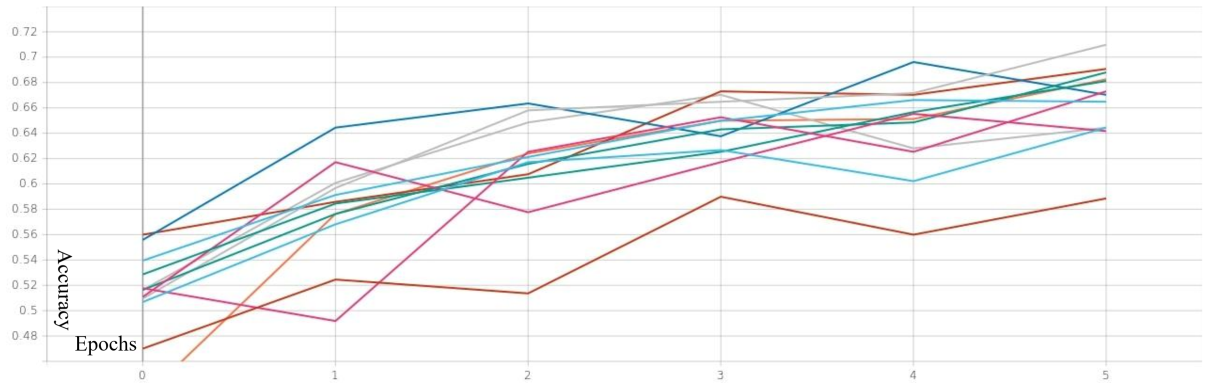


Epoch accuracy for the first generation

123



Epoch accuracy for the fourth generation



Epoch accuracy for the last generation

For this test we chose to showcast not only the last generation, but the second to last generation as well since it had the best accuracy. It is not unusual for the mean accuracy to lower sometime from one generation to the next, since the change of the offsprings might sometimes be in the wrong direction, but the idea is that over time, the change will still be a positive one (just how sometimes when training a NN, the epoch accuracy oscillates until it reaches its best value).

5. Conclusions and future work

This paper aimed to demonstrate the efficiency of using Genetic Algorithms for optimizing hyperparameters for Convolutional Neural Networks. To prove this, we compared the accuracy of a flower recognition CNN: initially, it ran using manual-implemented parameters and it reached an accuracy of 63%, then it ran using the GA to determine the parameters and architecture. Our tests reveal the impact that a GA has on the architecture and, consequently, on the accuracy of a CNN. Specifically, the tests indicate a rise in the accuracy of the flower recognition CNN, the numbers reaching 67.2% and, respectively, 66.8% in our second and third experiments. These results, put aside with the low number of generations and epochs that each CNN ran on due to the lack of resources of the used computers, show that even if ran for a short period of time or low number of generations, Genetic Algorithms have a strong impact on the accuracy and architecture. We predict that using Genetic Algorithms yields similar results if run on stronger computers, bigger convolutional networks and for a larger number of generations.

For future work, we consider that the GA used for our flower recognition CNN could be improved by experimenting with different parameter encodings (genes), a larger variety of selection and fitness functions and deciding which one indicates better results. On a larger scale, proving the importance of Genetic Algorithms could be supported also by running them on the CNNs showcased in the Choosing the network section, networks that are currently out of reach due to our lack of resources at the moment.

References

[1] Yanan Sun, Bing Xue, Mengjie Zhang and Gary G. Yen. *Evolving Deep Convolutional Neural Networks for Image Classification*. 2019. 1710.10741.pdf (arxiv.org)

[2] Edson D. Carvalho, Romuere R.V. Silva, Flavio H.D. Araújo, Ricardo, de A.L. Rabelo, Antonio Oseas de Carvalho Filho. *An approach to the classification of COVID-19 based on CT scans using convolutional features and genetic algorithms*. 2021. https://doi.org/10.1016/j.compbiomed.2021.104744

[3] Fabio R. Llorella, Gustavo Patow, Jose M. Azorin. *Convolutional neural networks and genetic algorithms for visual imagery classification.* 2020. https://doi.org/10.1007/s13246-020-00894-z

[4] Sehla Loussaief , Afef Abdelkrim. *Convolutional Neural Network Hyper-Parameters Optimization based on Genetic Algorithms*. (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 9, No. 10, 2018. Convolutional Neural Network Hyper-Parameters Optimization based on Genetic Algorithms (semanticscholar.org)

[5] Jason Brownlee. *How to Predict Sentiment From Movie Reviews Using Deep Learning* (Text Classification) (machinelearningmastery.com)

[6] Max Pooling. *Computer Science Wiki* [Interactiv] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling [cited: 2.1.2022]

[7] F. Johnson, A. Valderrama, C. Valle, B. Crawford, R. Soto and R. Ñanculef. *Automating Configuration of Convolutional Neural Network Hyperparameters Using Genetic Algorithm*. In *IEEE Access*, vol. 8, pp. 156139-156152, 2020, doi: 10.1109/ACCESS.2020.3019245. IEEE Xplore Full-Text PDF:

[8] Tensorflow. Flower Classification. classification.ipynb - Colaboratory (google.com)

[9] Nurshazlyn Mohd Aszemi, P.D.D Dominic. *Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms*. In IJACSA Vol. 10 No. 6 pp 269 2019. Hyperparameter Optimization in Convolutional Neural Network using Genetic Algorithms

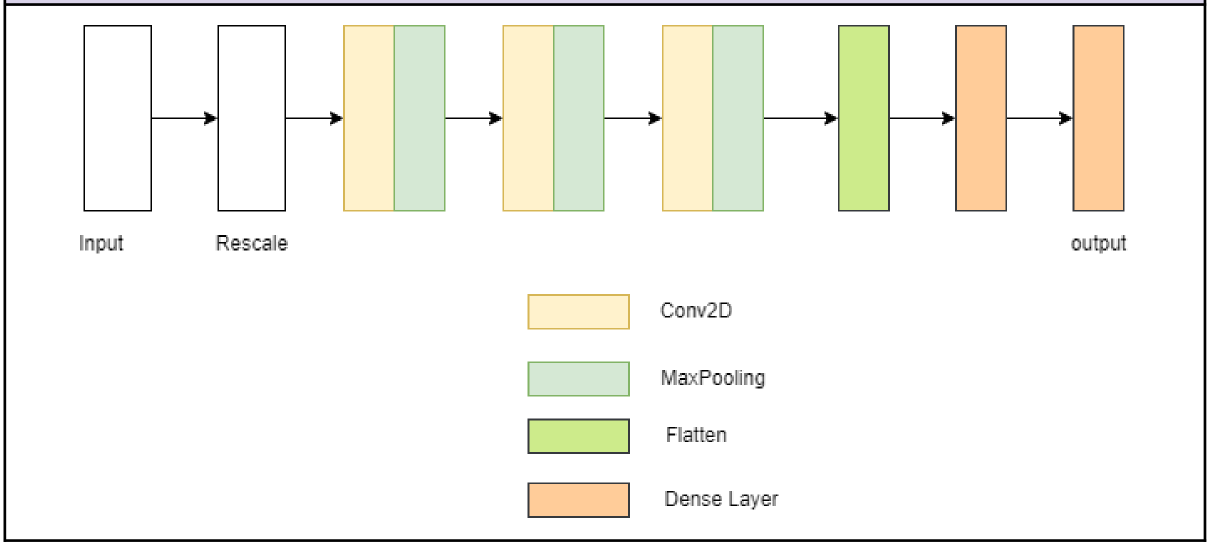
[10] Han, JH., Choi, DJ., Park, SU. *et al.* Hyperparameter Optimization Using a Genetic Algorithm Considering Verification Time in a Convolutional Neural Network. *J. Electr. Eng. Technol.* 15, 721–726 (2020). https://doi.org/10.1007/s42835-020-00343-7

[*] *Repository link: https://github.com/AndreiGS/GA_CNN.git*

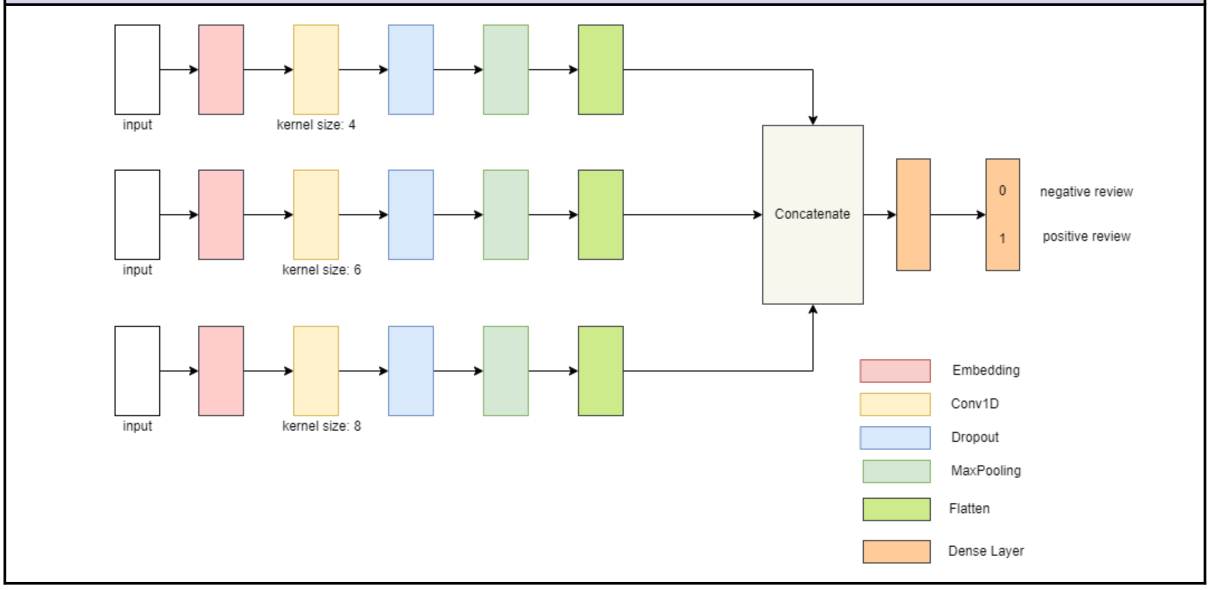
Appendices

Appendix A: Possible NNet analysis - detailed layers

1. *Flower Classification - the chosen CNN*

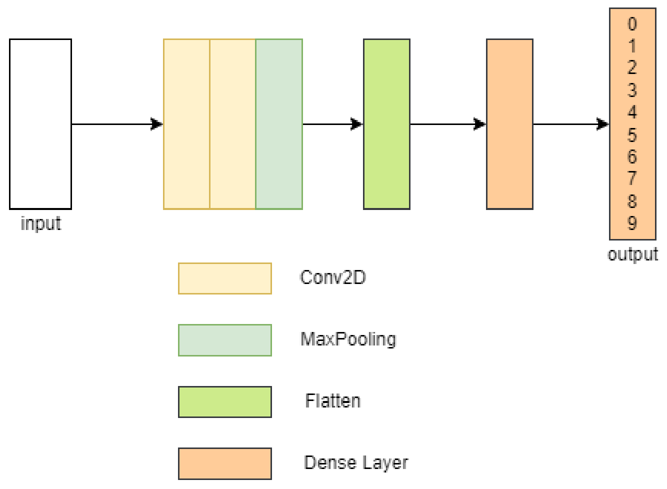


2. *Movie Review*

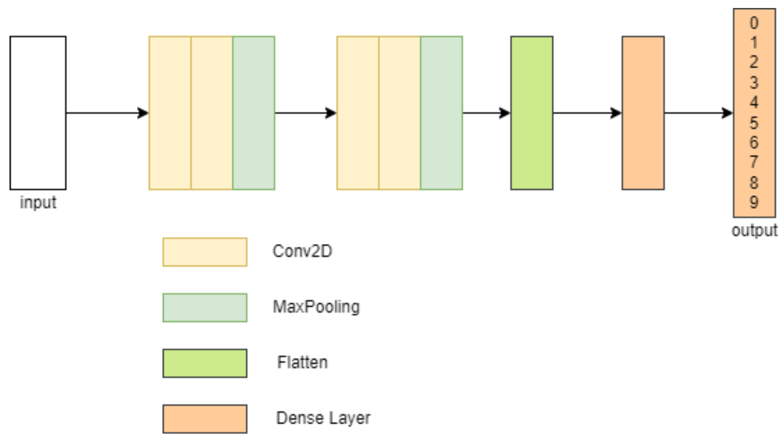


3. Cifar dataset

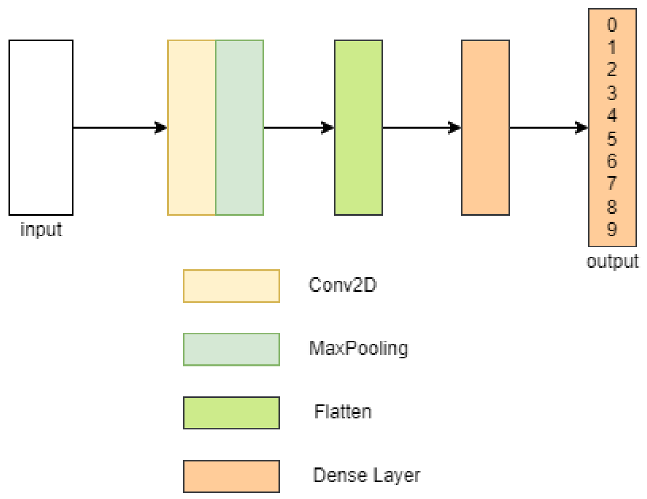
Model A:



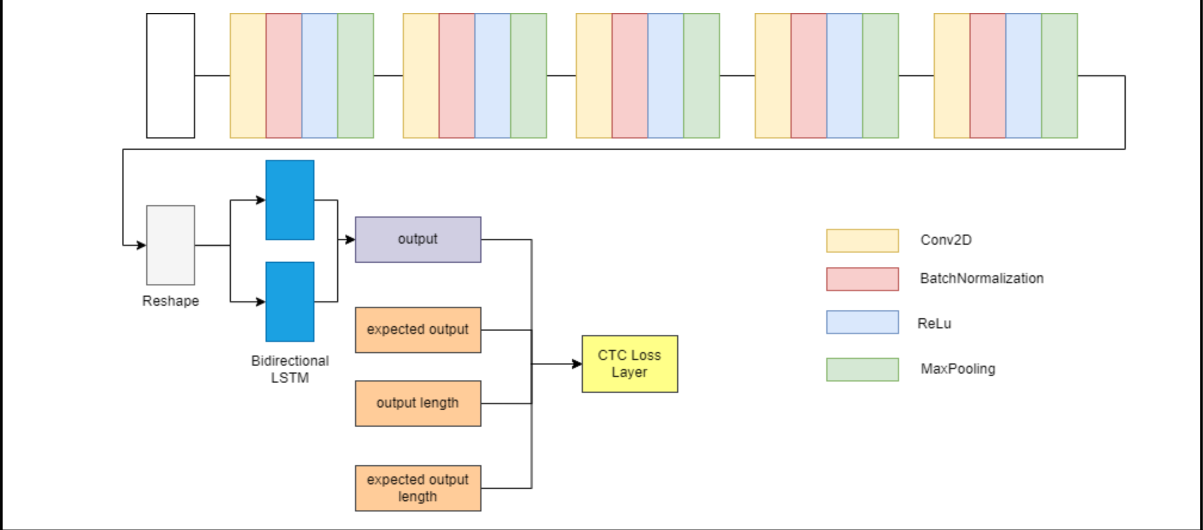
Model B:



4. Digit Classification



5. Handwritten text recognition


Reshape
Bidirectional LSTM
output
expected output
output length
expected output length
CTC Loss Layer
Conv2D
BatchNormalization
ReLu
MaxPooling