Hyperparameter optimization and architecture evolution in Convolutional Neural Networks for Image classification.

Korab Berisha, Hiroyuki Kido, Charith Perera

Abstract

Using genetic algorithms in order to autonomously produce CNN architecture-parameter combinations that provide good values in terms accuracy against large image data sets. The effect of different parts of CNN systems affect the efficacy of these networks differently, research into these areas led to the understanding as to the best choice of parameters and structure components to evolve, this was the general basis of all the inferential decisions made in this discussion. The results from these experiments showed that effective CNN architectures with optimized parameters were able tot be generated, however, the complexity of these systems and the ceiling of performance was degraded by the limitations placed upon the system by time and computational power.

Keywords: Hyperparameter, optimization, Neuro-structure, Neural Network, Convolutional, Convolve, Convolutional Neural Network

Contents

1. Introduction

Solutions to complex problems increasingly are being solved through the use of Neural Network (NN) structures. Neural networks are divided into many parts, every part is often manually selected in order to 1, decrease computational cost and 2, to increase efficacy of predictions in training models (Accuracy). For example, generating locomotive humanoids is a popular area of research in robotics. The generation of human robot movements with high fidelity is a frequently discussed area of research and garners various different approaches due to its emergence in the field. One such example, (Melo et al. 2021) [1] , used Deep Neural Networks (DNN) in an attempt to produce effective results in humanoid movement given the known lack of mathematical models. The hyper-parameters of these networks are often tuned manually given experience in the field and inferential understanding of artificial neural networks. The scale of neural networks can tend to the non tunable. This begs the question as to what methods are most effective in producing the best set of hyper parameters in neural networks, (Hutter et al. 2015) [2] investigates alternative methods of optimization.

Hyperparameter optimization involves finding the best set of values in the parameter space. Heuristic methods for finding said parameters relate to many topics including Data Science, Machine Learning (ML) and in the discussed evaluation, Artificial Neural Networks. Choosing the parameters has the ability to be autonomous but is often manual, however it remains intuition based in regards to manual selection. (Diaz et al. 2017) [3] . Recent years of research has divulged into many branches, some brute force, others Stochastic in the nature of their solution. For example, Random Search (RS), is frequently used in order to decide the most appropriate subset in the parameter space.

Genetic Algorithms (GA) are an example of a solution to Combinatorial Optimization solutions. It revolves around taking a problem where many solutions are available, and the selection of the best solutions. Research by (Alba and Tomassini 2002) [4] involved the discussion of evolutionary algorithms in optimization. Genetic algorithms make use of principle evolutionary processes. Survival of the fittest being one of the core components of GA. The generation of chromosomes is the building block of most GAs, as well as additional functions such as crossover, mutation and tournament selection used in the evolutionary process. The chromosome itself is the part that is generalized to arbitrary problems. This paper discusses the notion of using these chromosomes in order to find the global optimum in terms of hyper parameters and structure for the base CNN proposed.

2. Background

2.1. Convolutional Neural Networks

2.1.1. Convolutional Kernels and Feature Extraction

Convolutional Neural Networks are ANN's which are comprised of a chain or parallel chains of convolutional layers, pooling layers and fully

connected dense layers. Often visual computing tasks such as classification are tackled through Deep Convolutional Neural Networks because of their abstracting nature which performs well in analysis and classification of images (Lele 2018) [5] . Convolutional layers make up the majority of CNNs. The convolutional kernel can be considered as a matrix, and matrix multiplication occurs over the current input channel to produce the output for the next layer to process the information. Neurons in these layers have an output defined by the Activation Function (Apicella et al. 2021) [6] , for example, the Rectified Linear Unit (ReLU), comprises a simple equation $y = max(0, x)$.

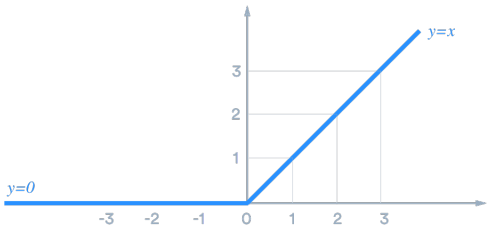


Figure 1:

CNNs have an ability to extract real features from visual data sets. This comes from the hierarchical structure of most CNNs. At the top, the number of filters is low, but as depth increases(more convolutional layers), Breadth (filter size) also increases, increasing the feature space but decreasing the spatial size. This allows the Feature extraction stage of CNNs to pick out small blobs and structures and slowly take those features and abstract them further. The more convolutional filters at the end means more input channels for the fully connected layer in the classification stage, this allows for weighted nodes to be associated with certain features, these nodes only activating when the correct combination of feature maps is presented. Therefore, classification can be achieved and accuracy increased adjusting these layers, values and so forth.

2.1.2. Pooling, Hidden Layers and classification

Pooling in CNN is used to address the problem of spatial complexity for images (Sharma and Mehra 2019) [7] . Pooling allows neural structures to simplify the values of the convolved feature. Typical construction of CNNs employ a number of convolutional layers with a pooling layer after in order to further reduce the data. How many layers, what kernel size and what method of pooling are some considerations when constructing CNN structures (Romanuke 2017) [8] . Relevant to the focus of this paper these values are hyper parameters as they need to be set prior to training and employing the model. In order for these structures to be resultant with valid predictions; this is achieved through a function of non-linear combinations in the feature space of the convolutional layers (Basha et al. 2020) [9] .

The Fully connected layer, similar to those in deep neural networks(N.R.E Et. al. 2021) [10] , make up these unseen layers and the data is fed forward, back-propagation is applied on every iteration of training. The classification ability comes from the abstraction of key features in certain categories. Activation functions such as softmax (Zhu et al. 2020) [11] .

2.2. Genetic Algorithms

2.2.1. Chromosomes: From Biology to Evolutionary Computing.

Much like the humane genome, genetic algorithms leverage the principles of darwininsm to achieve their goals, often encoded in binary bit strings (McCall 2005) [12] . The difference be-

tween binary chromosomes and protein based chromosomes is simply the goal of the overall system and the medium of existence. At the lowest level they are fundamentally similar given the fact they encode a series of instructions, a recipe for the creation of their product; Biologically inspired application to solution choice in large parameter space problems. Encoding takes the form of binary bit strings in most cases, however can be designed with its application use as the main consideration. Value encoding is another example of encoding in evolutionary algorithms. Value encoding is beneficial in scenarios in which the chromosome values relate to a specific set of data, i.e. {"tall","medium','short'} this is a data set which can be encoded as 1.5 binary bits, 0,1, and 00/11/01/10 and relates to a choice of heights for an arbitrary use. Exemplary of a situation in which simple binary encoding wouldn't be applicable due to the nature of the data and the limitations of choice, the secondary choice from the binary bits is necessary to overcome such issues.

2.2.2. Fitness

Fitness as a function of a single chromosome builds the foundation on to which the evolutionary algorithms progress in their desired direction(Alba and Tommassini 2002) [4] . As is seen in biology the fittest survive, and this concept is pivotal in GA. The fitness of a chromosome is decided by the needs of the algorithm, in the human genome this is the continuation of offspring and genetic lineage. This has its own secondary the fittest survive, and this concept is pivotal in GA. The fitness of a chromosome is decided by the needs of the algorithm, in the human genome this is the continuation of offspring and genetic lineage. This has its own secondary ner such that, the fitness can be manually set. A gene pool, or a set of chromosomes has a set of fitnesses attributed to said chromosomes. The highest fitness chromosomes are used as the template for future generations of populous.

2.2.3. Selection

The selection of parents in a gene pool is considerably important in the process of GA. Breeding occurs in order to continue the combination of the two selected genomes' lineage. This process is referred to as Selection (Alba and Tomassini 2002) [4] . Selection supports the process of siphoning the best combination of values. The higher the fitness of a gene the more likely it is to be propagated into the population over generations. Comparison of these approaches is researched by (Sharma et al. 2014) [13]. A range of many selection types are available all with their own drawbacks. The three selected for comparison are roulette wheel selection (fitness proportionate selection), tournament selection and elitism selection, considering all three the most compatible with the application will be selected.

2.2.4. Roulette Wheel Selection

A simple example of selection is the Roulette Wheel. It is based off of how real mating pools are proportioned with the fittest individuals taking up the bigger part of the population. Stochastic selection on a roulette wheel, with the fittest individuals having the biggest proportional section of the wheel (Lipowski and Lipowska 2012) [14]. It can be easily inferred as to how these fitter individuals would then be combined more frequently and therefore continuing the genetic code in the population.

Tournament Selection. Tournament selection differs from roulette wheel selection in the fact that it is not fitness proportionate. That meaning fitness does not have a correlation with selection probability. Tournament selection is done through selecting k-many random chromosomes, selecting the fittest out of the k chromosomes in a k-way tournament.

Elitism Selection. Elitism selection is more so a characteristic of the selection in itself. The method requires the generated populations chromosomes to be ordered by fitness. Then apply the selection process in that order, this way only fitness values one higher or lower than itself will be used for crossover as a parent. This means the genetic algorithm will breed strong genes with others, and weak genes the same [13] .

2.2.5. Recombination and Crossover

The process of crossover involves taking both parental chromosomes, and splitting them down the middle, opposite sides of both chromosomes result in the child.

Figure 2 clearly shows how single point crossover occurs between two selected parental chromosomes. This core aspect enables evolutionary progress bounded by the fitness metric. There are other methods of crossover, such as multi point, i which more than one point of intersection are selected. These points generally being selected at random between: 2 - genome size.

2.3. Optimization of Hyper-Parameters in ANN

Hyper parameters by definition are the variable attributes associated with a data science model, be it Machine Learning (ML) or ANN [2, 15] . Hyper parameters allow the designers of systems to fine tune their models in order to extract the best results. For example (Lele 2018) [5] , incorporated CNN in order to classify large data sets of images with various traffic signs. They used the AlexNet [15] CNN architecture to classify these images. The hyper-parameters associated with the specific convolutional neural network are: Size of Filter, Number of filters, Padding and Stride. The Size of the Filter is relevant to the network as it determines what ratio of abstraction occurs within the information pipeline. A larger difference between the input channel and the layer in front, results in a larger difference between the original kernel and the multiplied matrix. This has a resultant effect on the data and its level of reduction into features. The selection of these parameters however is not fixed in place and values must be selected for it. Different values resulting in varying efficaciousness. The Number of filters in a layer refers to the number of feature maps created by the respective filters, each filter creates its own feature map by being convoluted over the input. More filters results in more depth in the result. Padding is when, during the convolution operation per layer. Values are added along each edge of the current matrix in order to allow more involvement of edge pixels in feature mapping. Furthermore, stride is another hyper parameter involved in this operation if in use. The standard stride set automatically is often 1, meaning the convolutional filter is moved along the current matrix 1 pixel at a time, resulting in a larger feature map. A larger stride of 2 would result in fewer pixels used in the operation therefore a reduced sized feature map.

It is abundantly clear how this sort of parameter allocation can hugely impact the performance of neural network structure, however there are

tens of thousands of combinations depending on the network architecture. This raises the problem of how does one select the parameters for the optimum performance. i.e. what is the global maxima of fitness of networks given the parameter space.

2.3.1. Grid Search

Grid search involves using an exhaustive method of model hyper parameter tuning. A set of feasible hyper parameters is a prerequisite. Grid search is at its heart an optimization problem. Given a fixed set of values x, βresults in a value α. We can say α_{opt} is the optimum value as a function of $x$and β. $\alpha_{opt}(x\ \beta) = max_{\alpha}(\alpha\ x\ \beta)$ This mapping takes the parameters x,and β, and returns the validation accuracy (or any other fitness metric relevant). The problem is then states as such. Find the combination of values from the set of possible hyper parameters $x_{opt} \in \{x_1, x_2, x_3...x_n\}$ *and* $\beta_{opt} \in \{\beta_1, \beta_2, \beta_3...\beta_n\}$

Where $\alpha \approx \alpha_{opt}$ and the fitness metric is therefore maximized/minimized depending on the needs of the model. A brute force approach to search the parameter space is analogous to the grid search.

2.4. Network Architecture

In addition to the optimization of network parameters, these parameters come after the choice of a network architecture. The structure defined prior to the training of a neural network follows heuristic guidelines based on experience tuning these factors, and are often hand picked. CNN take the basic form of Convolutional Layers, where convolutional kernels iterate over the input image in order to make an abstracted feature map. The next part which most often follows Convolutional layers are pooling layers. Pooling layers [8] involve taking the feature map, and further reducing the spatial size in order to reduce computational cost. Localization of common features between individual images occurs between these layers, the further in depth the network progresses the more simplified the individual groupings of pixels become. Following these layers, prior to the final sector of a generic cov net, dropout layers are frequently used in order to prune neurons randomly to counteract over fitting, training accuracy of models is disproportionately high comparative to validation accuracy over a non-training data set in CNN without dropout layers between the pooling layer and fully connected layer [9] .

2.5. Datasets

Two data sets were selected in order to benchmark the proposed implementation. The first data set is the CIFAR-10 data set which contains 6000 x 32 images in 3 channels for colour per category, there are 10 categories. 50000 images are used in training and 10000 are used for validation. Similar to this is the CIFAR-100 dataset which has 100 classes with 600 images each class. 500 images rationed for training and 100 for validation. Both datasets have generally good performance with hand tuned CNNs. However, using the same cov net for both data sets would not work. This paper aims to show how automatic generation of network hyper parameters and network architecture can provide effective results over data sets.

3. Implementation

3.1. Overview

The implementation of the system designed made use of key machine learning libraries such

as tensorflow and keras. The concept suggested aims to provide a baseline CNN architecture, and develop combinations of possible architecture-parameter choices in order to find optimal local solutions.

3.1.1. CNN Generation

Feature Extraction Block. In order to allow CNN generation of different sizes and structures, a baseline feature extraction block was selected in order to be used as a building block of the CNN. This consisted of a 2 dimensional convolutional layer with a argument based number of filters, the stride of these layers is set to 1. This was decided through testing and evaluation of large sets of generated models with the stride as the dependent variable. A smaller stride increased computational cost however improved performance so it was chosen as part of the baseline selection. An activation function is used on the output for the conv layer in order to calculate activation probability and further add a bias to the weight in order to either increase or decrease likelihood of neuronal activation based off of the effectiveness of the direction of change in the weight.

```
1 def conv_layergen(model,ksize,
     inputsize,acti):
2     model.add(layers.Conv2D(
          inputsize, (ksize, ksize)
          ,padding="SAME",strides
          =1))
3     model.add(layers.Activation(
          acti))
4     model.add(layers.
          MaxPooling2D((3, 3),
          padding="SAME"))
5
6     return model
```

Code Snippet 1: Feature extraction block generation

Test and Train dataset import and normalization.. The cifar10 and cifar100 datasets included in the keras [16] data sets are downloaded using tensorflow and the images for training and testing are separated, this set of data is then normalized by dividing the images by 255.0, the floating-point notation of the brightness value of a pixel.

```
1 mnist = tf.keras.datasets.mnist
2 (train_images, train_labels), (
     test_images, test_labels) =
     datasets.cifar100.load_data()
3 train_images, test_images =
     train_images / 255.0,
     test_images / 255.0
```

Code Snippet 2: Dataset loading, and normalization

CNN Generation from binary encoding selection. The model_gen method allocates several arrays in which the encoding in the binary bitstring correlates to. For example characters 0-2 in the bitstring will be two bits, converted to an integer it is used as the index in the selection choices and thus whole bitstrings represent whole combinations of architecture and hyper-parameters. The parameter space for this model contains both.

The sequential model method from Tensorflow [17] initializes the stack of layers. At the addition of every layer in the CNN the model is accessed and the add method is called in order to insert a new layer in the model stack. After the input of the model, in the size of the dataset image resolution, in this case 2 dimensions of 32 spatial pixels and 1 dimension of 3 colour channels (32x32x3). After this the loop dictating the depth

of the speciifed model begins, where the depth of the model is the layer_levels array content at the index given by the integer value of the encoded binary subset. The loop adds a feature extraction block of filter size content at index conv_selection in output_options. This same loop is repeated by the number of feature extraction points specified again by another subset of the encoded chromosome. Finally, a dropout layer with the value specified by the choice in dropout options is added after every feature extraction layer in order to reduce overfitting in certain cases.

```
1  for 1 to layer_levels[
       conv_selection]
2    if convlayers_feature[0]
         == "1":
3      add 1 layer with 1
         conv layer to
         model with
         parameters
4    else:
5      add 1 layer with 2
         conv layers to
         model with
         parameters
```

Code Snippet 3: Model generation based off of chromosome encoding (pesudocode)

The feature extraction loop adds ends after the final one is added to the layer stack, the entire n x m x d array is then flattened into a 1 dimensional vector in order to be used in classification. The implementation of fully connected layers allows for high level features within the feature space of the network to be used as the inputs for the classification capabilities given by fully connected dense networks. These high level features are the indicators to the dense layers as to what the image is. The final dense layer is set at the number of classes in the dataset (100)

```
1  model.add(layers.Flatten())
2  model.add(Dense(shp[1]*2,
       activation="relu"))
3  model.add(layers.Dropout
       (0.5))
4  model.add(Dense(shp[1],
       activation="relu"))
5  model.add(layers.Dropout
       (0.5))
6  model.add(Dense(100,
       activation='softmax'))
```

Code Snippet 4: Fully Connected section, classification from fe

3.2. CNN Architecture

3.3. Binary Encoding Scheme of Hyperparameter chromosomes

The computational cost of GAs is dependent on many factors, including the mutation rate, choice of selection method, population size and number of generations. These are tunable and selected based of intuitive reasoning, observation on performance is the easiest way this is done. The final parameter of GA that affects performance is the genome size. The genome size represents the information the algorithm is aiming to evolve to optimal fitness [12]. Therefore, the number of items is limited by the length of the binary encoding. A 20 bit binary string is used in order to represent the information regarding the hyperparameters and network structure. Every category of hyperparameter has a pre-defined number of choices.

The encoding of these values represents the actuality of the CNN architecture and combinations

of hyper parameters [18] . For example a chromosome of 0110010010110 would represent an CNN with 28 filters in the first feature extraction section, the filter scale of 3 implies the number of filters will be scaled at every layer proportionally to the hierarchy and number of inputs. So for example if there was 3 feature extraction stages, the first value would be 28, the selected value, but it would be multiplied by the filter scale, in order to increase the depth, the width has to also increase to accommodate more features. 16*3,32*3,64*3 = 48, 96, 192. Following this the next bit is representative of the kernel size, this affects the neural network heavily due to the varying levels of abstraction produced by different sized kernels, a larger kernel produces a smaller feature map, losing data but generalizing edges blobs and possibly features to specific areas in categories. The number of feature maps is stated by the 13th - 19th bits. Every bit represented in this subset represents every possible feature map in the network, these bits themselves dictate the number of convolutional layers per feature map, if the bit is a 1, then the number of layers is one, if it is 0, then the number of layers is two. Initially i attempted to add more variation in terms of number of conv layers per feature map stage however the number of parameters in the network was high enough to make the entire process redundant due to the time needed to process multiple sets of multiple networks; In combination with the fact performance did not drastically increase near the ceiling of the GA fitness rankings.

3.4. Genetic Evolution of Hyperparameters

There are many issues of varying importance when considering the autonomous selection of hyper parameters in CNN's. If the input shape of a convolutional network is 32x32x3, with a kernel size of 3, a stride of 2, and pooling size of 3x3, with 32 filters, the output shape would be 11x11x32. This is an example of a feature extraction block in many CNN architectures(with varied parameters). This is applicable to many problems through human construction, however when considering an autonomous alternative it seems intuitive to limit options to those that would result in a more effective trained model, or an individual in the gene pool contextually. In order to overcome the issues regarding spatial size of convolutional kernels, padding is often used in order to allow kernel sizes not factor able into the spatial size to be used, in combination with only allowing a certain range of kernel size therefore reduces the size of the possible parameter space.

Reducing the time taken to find fit neural networks, the number of epochs was kept low, this has its effect accuracy, however as it is shown in the results, I found early epochs were generally a good sign of fitness looking ahead. The specific value chosen is only bounded by 1- the accuracy of the future depiction of the network based of early epoch values, and 2- the amount of time the user has available to themselves.

The generation of CNN using this technique allows us to generate likely good candidates no matter the combination of choices, this is done to reduce the total number of possible combinations, and more importantly reducing the number of combinations to those whose content is viable in the context of accurate networks. For example a CNN generated by the GA which achieved a 76% validation accuracy on the CIFAR-10 dataset is illustrated in Figure 3 and in Figure 4 as a tensor diagram.

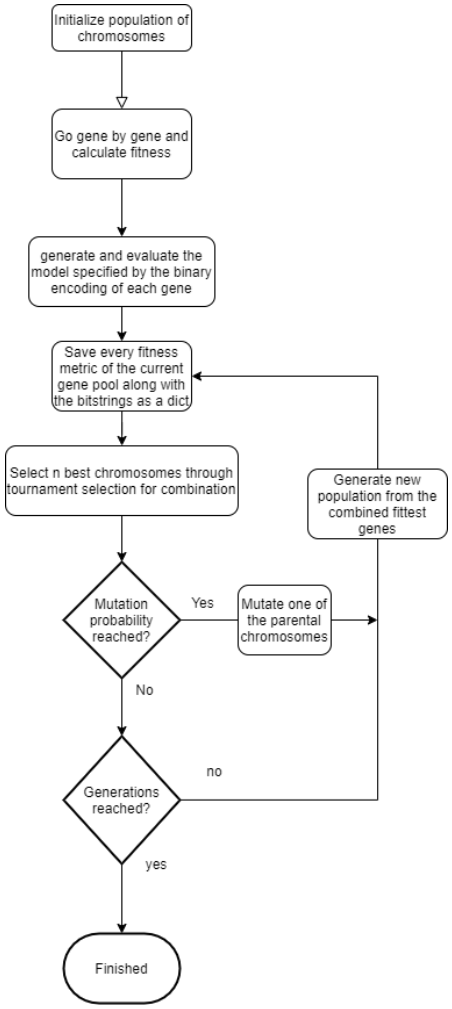


Figure 2: CNNGA Flowchart

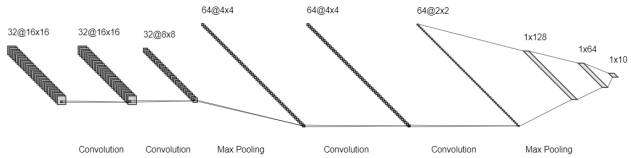


Figure 3: CNN from chromosome "01010001110010011100"

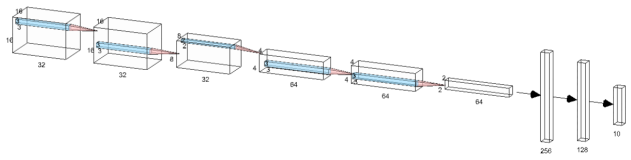
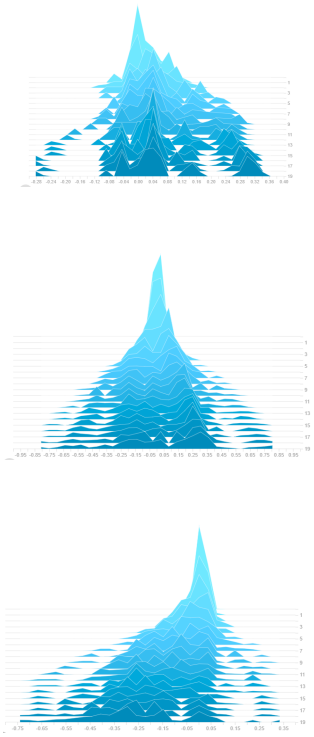
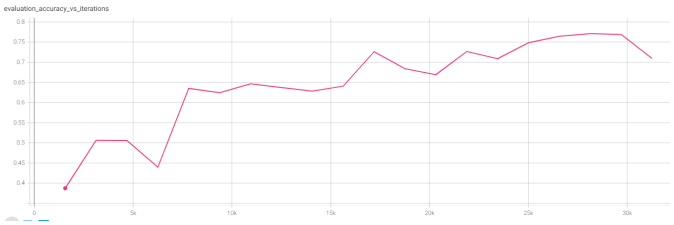


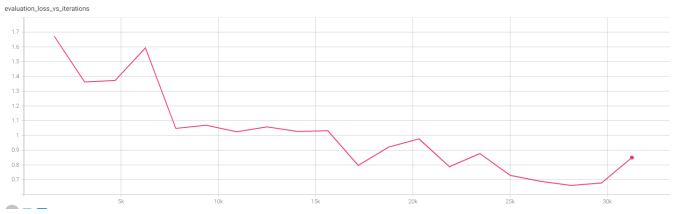
Figure 4: CNN from chromosome "01010001110010011100" tensor blocks

As an inductive example, this singular CNN classifys images effectively, it can be seen learning occurs between every stage of convolution. This is suggested by taking contextual data between convolutions at every iteration and comparing bias and weight with their respective selves.



These three histograms representing three of the convolutional layers and their bias against the number of epochs the network has trained on. An example network this information was used to ensure that networks generated remained functioning at the bottom end of the gene pool. It can be seen as the epochs progress through the first layer, there are many changes to the distribution of the bias, which can be inferred to mean something is being learned, it is safe to assume those items being learned are features extracted from the image set as the validation accuracy of this network was high and proved it was an effective classifier.





It can be seen that this network achieved a validation accuracy of almost 78%, the training accuracy was slightly higher at 81% however this can be attributed to over fitting due to possible lack of testing data diversity, or similarly due to an abundance of neurons which are detrimental to the classification and feature extraction processes respectively. This methodology was used to ensure the base structure and variants of said structure would result in trainable networks. Given this understanding i proceeded with the experiment in order to ascertain if said development of networks was applicable in overall network optimization.

4. Results and Issues

In order to justify the usage of genetic methods this experimental approach aims to provide a distinct usage for this solution over multiple data sets and analytical understanding of individual results in order to reduce the likelihood of results which lack replicability.

Genetic Parameters Used:

- Population Size = 4
- Genome Size = 20
- Generations = 20
- Mutation Factor = 1 / Population Size

Base CNN Architecture/Parameters:

- Pooling Size = (2 x 2) [For all pooling layers]
- Padding = Same (0 padding)
- Three Hidden Layers consisting of:
  - 2048 Fully-Connected neurons with a 0.5 dropout rate and ReLU activation layer proceeding
  - 1024 Fully-Connected neurons with a 0.5 dropout rate and ReLU activation layer proceeding
  - 10/100 (CIFAR-10/CIFAR-100) Full-connected neurons with a SoftMax activation function
- Batch Normalization at every feature extraction stage.

Running the software for around 2 hours with these values set gives us 3 generations of network combinations tested. The graph below shows how as time steps, each networks values in terms of validation accuracy.

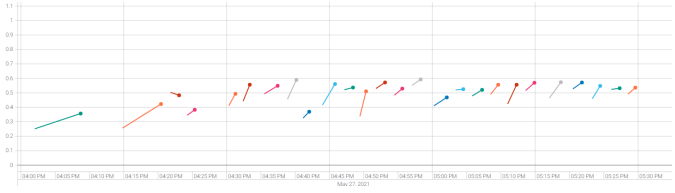


Figure 5: Generational itterations of different networks

As is visible in Figure 5 each line represents a singular CNN given by the current fitness test on the current gene. The usage of two epochs allows the system to run for an extended period of time without taking an amount of time which would be contextually erroneous. As is observationally obvious, the variation in results reduces as time goes forward, this indicates the genetic variation between the whole population is being reduced, to that of values which are consistently high/similar to other individuals in populous which have been propagated through the genome. The chromosome "10000000010100001000", which had the highest validation accuracy of the genomes within the third generation is taken as an example and will be run for 20 epochs in order to find its global optimum in terms of iterations run. Breaking this chromosome down into the network representative, we find the first two bits relaying infromation regarding the number of filters, most chromosomes in the population end up with the same two beggining bits that being 10. Given the set of chromosomes in the third generation.

Table 1: Gen 3 of CIFAR-10 Evolution

Chromosome	Validation Accuaracy
10000000010100001000	0.5455999970436096
10000000000010010000	0.5648000240325928
10000010000000010000	0.5328999757766724
10000000000010000000	0.5734999775886536

The information in Table 1 distinctly show the gene pool is very similar, implying the fitter portions of the chromosome have been passed down through parental reproduction of the fittest individuals.

Figure 6 indicates that this particular chromosome produces a network which has good distance from actual values, 0.8 validation loss. However, it can be inferred that the variance between the two plots is indicative of under fitting; This further clarifies that there is room for improvement given a higher number of epochs in the network training regiment.

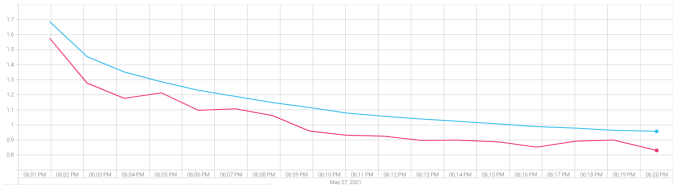


Figure 6: Training vs Testing Loss value per epoch

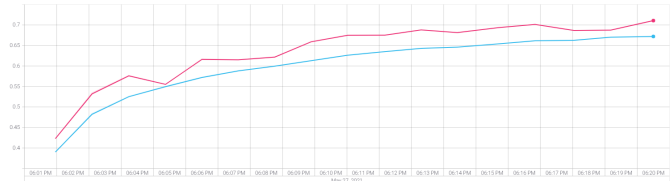


Figure 7: Training vs Testing Accuracy per epoch

Figure 7 shows how a fit individual from an early generation provides a good combination of structure and hyper parameter choices, this is inferred from good results in accuracy scores. Accuracy scores of 78% were achieved on the testing data. in comparison, with high level deep convolution networks (Krizhevsky et al. 2017) [19] achieved accuracy ratings of over 89% against the testing set of images on the same data set. This is not to indicate that the method involved in producing the results in Figure 7, it is entirely possible that the increase in epochs would produce even closer results.

Considering what the issue in the results may be, the bias in terms of every convolutional layers weights overlayed over time in order to make a comparative assessment as to what the issue could be.

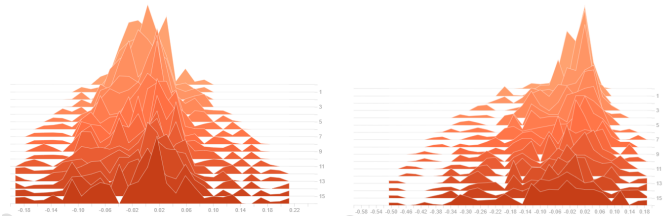


Figure 8: Feature extraction layer one

Figure 8 shows the bias of the first two (begginning on the left) convolutional layers in the first of three feature extraction sections in the CNN. It is observed that in both graphs, the distribution of values decreases in range but increases in variance as epochs increase. It can be assumed this layer would most likely extracts effective features.

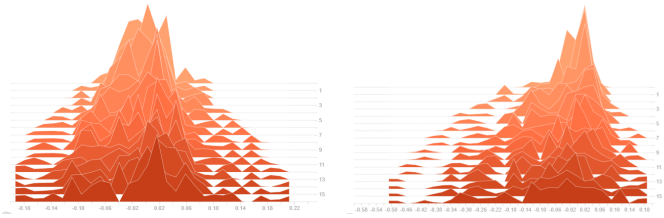


Figure 9: Feature extraction layer two

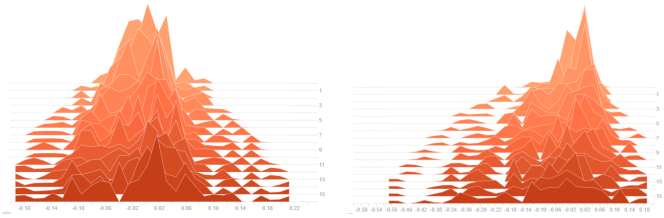


Figure 10: Feature extraction layer three

Figure 9 and Figure 10 both continue down this same path of decreasing range and increasing variance. Influentially it can be shown that the peaks in the final epoch are representative of low level features in the image space. These feature maps can be effective as it is visible as to specific areas on the plot, where bias is low, it is extremely low, almost nill; whereas where bias is high, it is very high, in respect to the rest of the peaks, these peaks being either too high or too low results in feature being over or under prioritized based on the features impact on classification itself.

5. Conclusion

Hyperparameter and CNN structure optimization remains a sough after research field. Many issues and problems awaiting resolutions exist. In conclusion, there are many different classification systems and the creation of them is still a disputed topic within computer science. It can be said the results in this paper indicate that there is a strong sign progression of network performance as generations increase in the GA system. It can be seen in the results that these autonomously generated CNN models are effective feature extractors and classifiers. The entire conception of this system was inspired by related works and aimed to reduce the computational cost normally associated with these highly demanding systems. Pruning ineffective parameters in order to allow the system to find its ideal choice. It can be said that the GAs Mutation operator does not make much of an impact due to the lack of ability to generalise the system to more generations. In conclusion, many network architectures such as AlexNet perform similarly to the GA system proposed here, however the key difference is the reduced need for human intervention to create the CNN.

6. Reflection and Improvements

6.1. Changes during development

There are countless changes from my initial proposal to my end result. The tuning and development of the system and the fundamental concepts driving it was not linear, changes made sometimes influenced the efficacy of the system negatively

6.2. *Future work and why these issues remained during development*

The methodology discussed in this paper has boundless opportunity given unlimited time and computational capacity. However, the limitations induced by both of these factors heavily influenced the quality of the results and the breadth of information inferred from the experimentation.

In combination with these issues. Applying this these techniques to harder data sets such as CIFAR-100 came with many more challenges, in terms of genetic search and finding optimal structures, the GA proved itself relatively effective. However, the application to CIFAR-100 found the software taking many days to only run one or two populations worth of neural networks. The use of parallel or distributed systems is one solution to this issue and In retrospect, the project would have more statistical data in order to back up proposed solutions in the autonomous generation of CNN systems for image classification in the scenario where high performance computer systems were used.

In an attempt to apply the same system proposed to the harder CIFAR-100 dataset, a simple change of the number of classes available in the dataset used as the number of neurons in the final fully connected layer in order to classify.

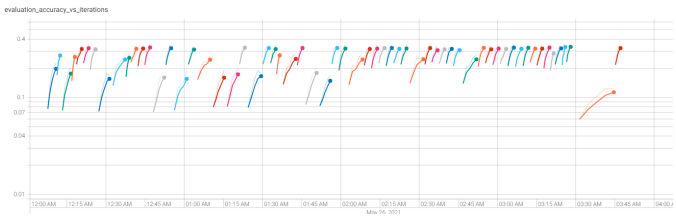


Figure 11: 10 Generations of CNNs training and testing on CIFAR-100

This graph shows roughly 60 separate CNN architecture and parameter combinations along the time axis. The number of epochs used in this case was 5 as if the system used 2 epochs such as that in the CIFAR-10 set as an early marker for fitness, it would be difficult to differentiate good and bad systems as they are often varied at a high degree in the first epochs.

One of the better networks from this set was used as a sample, the network was tested over 50 epochs instead of 5, and the results are visible below.

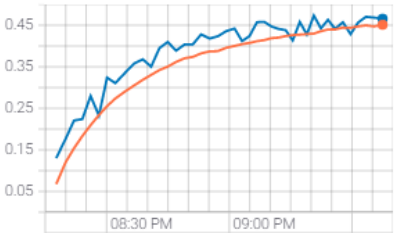


Figure 12: Accuracy vs Validation accuracy against the CIFAR-100 Dataset

Achieving an accuracy of 0.45 is sub optimal in terms of many state-of-the-art systems, however the information garnered from these statistics show that given more time to run the genetic algorithm, even better networks would be found. Furthermore, running the networks for a longer period of time would in turn increase the accuracy.

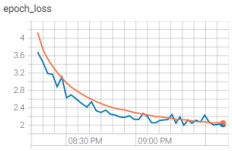


Figure 13: Loss vs Validation loss against the CIFAR-100 Dataset

In the loss chart it is clear that the training and testing accuracy equalize. This is indicative of the network reaching its potential, as if the loss value plateaus then we know the network is not improving in accuracy, but furthermore the predictions

being made are consistently wrong, and the degree of error between predictions is not reducing. The figures Figure 12 and Figure 13 indicates little over fitting occurring in the experiment. The deviation between the values in both fields remains very small.

Improvements upon the system would be limitless if constraints were non-existent, many alternative and less impact full parameters could have been implemented in order to fully find the optimal set of choices. Furthermore, the depth and breadth limits imposed by the selection choice were implemented to further reduce computational cost, however undoubtedly would increase the selection space to that where even better combinations of choices could possibly exist.

References

[1] M. Melo, A. D. Maximo, Cunha (2021). [link].
URL https://arxiv.org/abs/1901.00270

[2] F. Hutter, Beyond Manual Tuning of Hyperparameters, KI - Künstliche Intelligenz 29 (4) (2015) 329–337.

[3] G. Diaz.

[4] E. Alba, M. Tomassini, Parallelism and evolutionary algorithms, IEEE Transactions on Evolutionary Computation 6 (5) (2002) 443–462. doi:10.1109/tevc.2002.800880.
URL https://dx.doi.org/10.1109/tevc.2002.800880

[5] N. S. Lele, Image Classification Using Convolutional Neural Network, International Journal of Scientific Research in Computer Science and Engineering 6 (3) (2018) 22–26. doi:10.26438/ijsrcse/v6i3.2226.
URL https://dx.doi.org/10.26438/ijsrcse/v6i3.2226

[6] A. Apicella, A survey on modern trainable activation functions", Neural Networks 138 (2021) 14–32.

[7] S. Sharma, R. Mehra (2019). [link].
URL https://dx.doi.org/10.2478/fcds-2019-0016

[8] V. V. Romanuke, Appropriate Number of Standard 2 × 2 Max Pooling Layers and Their Allocation in Convolutional Neural Networks for Diverse and Heterogeneous Datasets, Information Technology and Management Science 20 (1) (2017). doi:10.1515/itms-2017-0002.
URL https://dx.doi.org/10.1515/itms-2017-0002

[9] S. Basha, Impact of fully connected layers on performance of convolutional neural networks for image classification, Neurocomputing 378 (2020) 112–119.

[10] N. Et, Feature Extraction In Gene Expression Dataset Using Multilayer Perceptron", Turkish Journal of Computer and Mathematics Education (TURCOMAT) 12 (2) (2021) 3069–3076.

[11] Q. Zhu, Improving Classification Performance of Softmax Loss Function Based on Scalable Batch-Normalization", Applied Sciences 10 (8) (2020) 2950–2950.

[12] J. Mccall, Genetic algorithms for modelling and optimisation, Journal of Computational and Applied Mathematics 184 (1) (2005) 205–222.

[13] P. Sharma, Analysis of Selection Schemes for Solving an Optimization Problem in Genetic Algorithm", International Journal of Computer Applications 93 (11) (2014) 1–3.

[14] A. Lipowski, D. Lipowska, Roulette-wheel selection via stochastic acceptance, Physica A: Statistical Mechanics and its Applications 391 (6) (2012) 2193–2196. doi:10.1016/j.physa.2011.12.004.
URL https://dx.doi.org/10.1016/j.physa.2011.12.004

[15] S. Samir (2020).

[16] K. Team (2021). [link].

URL https://keras.io

[17] Tensorflow (2021). [link].
URL https://www.tensorflow.org

[18] M. Wistuba, . Berlingerio, M, Deep Learning Architecture Search by Neuro-Cell-Based Evolution with Function-Preserving Mutations, Machine Learning and Knowledge Discovery in Databases. ECML PKDD 11052 (2018).

[19] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet classification with deep convolutional neural networks, Communications of the ACM 60 (6) (2017) 84–90. doi:10.1145/3065386.
URL https://dx.doi.org/10.1145/3065386