Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classification

Reeti Pandey, MEng Artificial Intelligence, University of Cincinnati
Siva Durga Abhinay Karrothu, MEng Artificial Intelligence, University of Cincinnati

Abstract— **Deep Learning is emerging to be the most powerful technique in solving many modern-day problems on Image, Text, Voice, Signals and Statistical data. The success of Deep Learning models can be assessed from the various state-of-the-art pretrained models available right now like those developed by Google and Microsoft. These models are developed after years of research and experimentation. For someone who builds a model from scratch, attaining high levels of accuracy is a time intensive and iterative process. Also, it also requires a high level of domain expertise and in-depth understanding of Deep learning models to be able to build a state-of-the-art accurate model. Researchers have been doing efforts to come up with ways to automate this process of configuring deep learning parameters. Usual methods of brute force search like grid search for hyperparameters will not succeed in this case given the computational complexity of Deep Learning models. This calls for an intelligent and adaptive algorithm that takes over the task for hyperparameters configuration with minimal intervention and supervision. Evolutionary algorithm also known as genetic algorithm has been around for quite some time. It is popularly used in many multi-objective optimization problems. Our research shows that it is particularly fruitful in the case of hyperparameter configuration for Deep Learning models as it reduces the search space compared to grid search and is extremely time efficient.**

Keywords— Image Classification, Genetic Algorithm, Evolutionary Algorithm, Convolutional neural networks, genetic algorithms, neural network architecture optimization, evolutionary deep learning.

I. Introduction

Convolutional Neural Networks are hugely popular owing to their computational simplicity and efficiency in almost all kinds of computer vision problems, ranging from image classification to image detection. t has been known that the performance of CNNs highly relies upon their architectures. To achieve promising performance, the architectures of state-of-the-art CNNs, such as GoogleNet, ResNet and DenseNet, are all manually designed by experts who have rich domain knowledge from both investi- gated data and CNNs. Unfortunately, such domain knowledge is not necessarily held by each interested user. For example, users who are familiar with the data at hand do not necessarily have the experience in designing the architectures of CNNs, and vice versa. As a result, there is a surge of interest in automating the design of CNN architectures, allowing tuning skills of the CNN architectures to be transparent to the users who have no domain knowledge on CNNs.

Evolutionary algorithms have been used to solve multi objective optimization problems for decades. Hyperparameter optimization problem is also a multi objective optimization problem in the case of CNN's. There are a wide range of parameters that can be altered and the model performance change drastically. However, there is no universal recipe of optimum combination of these parameters. A variety of combinations are to be tried based on intuition and prior domain knowledge in order to arrive at the best suited model. The choice of parameters is also a function of the type of problem we are solving like image classification or image enhancement or object detection and so on. Given the number of variables in picture, it becomes next to impossible for a beginner or novice to build an accurate CNN model for a dataset at hand. Even for advanced level learners, rewriting and reconfiguring codes from one problem to another require significant efforts as the CNN architecture demands fine tuning and sometimes rearranging as per the problem statement and type of data.

This lays the foundation for an algorithm that can perform the task of configuring CNN parameters with minimal supervision, with limited computational resources and in stipulated amount of time. Traditional grid-search like algorithms isn't successful in case of deep learning models because the search space is large and run time is huge for each iteration. Hence, the need for intelligent and adaptive algorithm that can search for optimal solution within limited resources and time.

Genetic algorithms leverage the principle of natural process of evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation which is analogous to hyperparameters in neural network. Best set of hyperparameters are chosen for mutation and their variants further mutate to ultimately arrive at the optimal set of hyperparameters. Gene/offspring here is symbolic to the sequence of hyperparameters used by CNN. Mutation is analogous to the variation in hyperparameters to some extent while keeping them somewhat similar to the parent gene. Fittest individual is decided by the validation accuracy of the CNN model over the chosen dataset for the chosen problem. The amount of noise that mutation and random generation of parent gene adds to the process ensures that every run of the evolutionary algorithm gives different optimal result.

Overall, the process of using Genetic algorithm to select optimal CNN hyperparameters also gives as a byproduct, some understanding around the relationship between the model hypermeters and performance of model.

II. BACKGROUND

In 2014, Dr. Islam, Dr. Badar Baharudin, Zuhairi, Raza, Muhammad Qamar & Nallagownden, Perumal worked on Optimizing a neural network architecture using genetic algorithm for load forecasting. Their work included using a genetic algorithm to find the optimal number of neurons in the input and hidden layers. Even though they optimized a basic neural network that is ANN they achieved a better accurate result in forecasting and with their work opened a new phase in building deep learning algorithms using Genetic Algorithm.

In 2020, Dr. Pham and Dr. Tran Designed a complete deep neural network using a genetic algorithm for estimating pile bearing capacity in which they aimed at deep learning neural network that selects optimal parameters for the Deep Learning Neural Network, including network architecture, activation functions for the hidden layers, number of hidden layers and number of neurons in each hidden layer by a genetic algorithm. In this they used genetic algorithm to select more complicated parameters for a deep learning algorithm.

In the same year, Dr. Mustafa abbas, Dr. Sabrina Tuin and their team optimized extreme learning machine approach using genetic algorithm for automatic COVID-19 detection in which chest X ray images are classified into healthy and COVID-19 infected.

III. METHODS

In this section we present the detailed overview of the framework of the proposed algorithm. The two broad sections of the framework are the Deep Learning pipeline, we will be referring it as DL framework and second is Evolutionary Algorithm framework, we will be referring it as the EA framework. The EA framework supplies the DL framework with sequence of model parameters that are parsed and used by the DL framework for training the CNN model for fixed number of epochs and testing the learnt model on validation set.

The EA framework has two major functions, one for initialization of parent gene sequence, which is the ordered list of model parameters for DL framework. The other function is for mutation of parent gene with a mutation probability ‘r’. Both functions generate a gene sequence i.e., ordered list of model parameters. This ordered list is passed to the DL framework. DL framework then parses it into model parameters for CNN and build the CNN architecture. This model is trained for fixed number of epochs and tested over a validation set. The accuracy received on validation set is then compared against the best accuracy seen so far. If the model outperforms the best seen model so far, it is saved to the memory as well as the model parameters are returned as the best model and the accuracy is returned as the best accuracy.

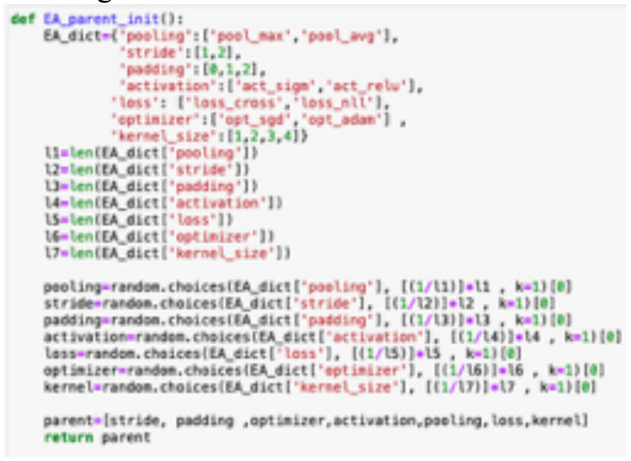
The driver module keeps track of all the iterations performed and the model parameters/gene sequence passed in each iteration. It also maintains a tabular record of the accuracy of model generated in each iteration and the best model and accuracy seen so far.

3.1 PARENT INITIALIZATION MODULE

This module is responsible for generating a parent gene sequence which is an ordered list of model parameters. It selects one of the alternatives for each model parameter from a dictionary of parameter values. An alternative is chosen with an equal probability given to each instance.

Algorithm Overview

Input: Dictionary of parameter names and their available alternatives.
output: List of parameters chosen at random from available alternatives.
Gene ← pick one instance from all available alternatives at random.
Return gene



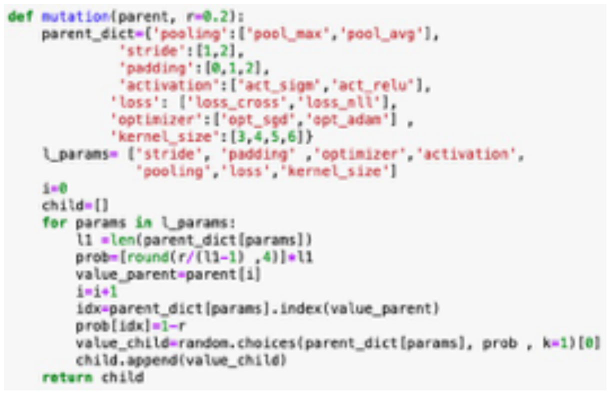
```
def EA_parent_init():
    EA_dict={'pooling':['pool_max','pool_avg'],
             'stride':[1,2],
             'padding':[0,1,2],
             'activation':['act_sigm','act_relu'],
            'loss': ['loss_cross','loss_nll'],
            'optimizer':['opt_sgd','opt_adam'] ,
            'kernel_size':[1,2,3,4]}
    l1=len(EA_dict['pooling'])
    l2=len(EA_dict['stride'])
    l3=len(EA_dict['padding'])
    l4=len(EA_dict['activation'])
    l5=len(EA_dict['loss'])
    l6=len(EA_dict['optimizer'])
    l7=len(EA_dict['kernel_size'])

    pooling=random.choices(EA_dict['pooling'], [(1/l1)]*l1 , k=1)[0]
    stride=random.choices(EA_dict['stride'], [(1/l2)]*l2 , k=1)[0]
    padding=random.choices(EA_dict['padding'], [(1/l3)]*l3 , k=1)[0]
    activation=random.choices(EA_dict['activation'], [(1/l4)]*l4 , k=1)[0]
    loss=random.choices(EA_dict['loss'], [(1/l5)]*l5 , k=1)[0]
    optimizer=random.choices(EA_dict['optimizer'], [(1/l6)]*l6 , k=1)[0]
    kernel=random.choices(EA_dict['kernel_size'], [(1/l7)]*l7 , k=1)[0]

    parent=[stride, padding ,optimizer,activation,pooling,loss,kernel]
    return parent
```

3.2 Mutation

This module is responsible for generating the mutated child gene sequence from the parent gene sequence with a mutation probability ‘r’. It does so by giving a higher preference (1-r) to the alternative chosen by the parent gene sequence and a different lower preference to all other alternatives.

Algorithm Overview

Input: Sequence of model parameter of parent and mutation probability, ‘r’, by default it is kept as 0.2.
output: List of parameters chosen from available alternatives by giving higher preference (1-r) to alternative chosen by parent and equal lower probability (r/n-1) to rest of the alternatives. Here ‘n’ is total number of available choices for the parameter.
Child← pick one instance from all available alternatives with the probabilities defined earlier.
Return child



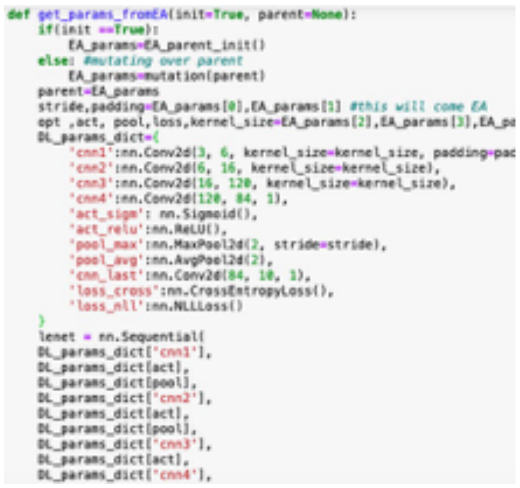
```python
def mutation(parent, r=0.2):
    parent_dict={'pooling':['pool_max','pool_avg'],
             'stride':[1,2],
             'padding':[0,1,2],
             'activation':['act_sigm','act_relu'],
            'loss': ['loss_cross','loss_nll'],
            'optimizer':['opt_sgd','opt_adam'] ,
            'kernel_size':[3,4,5,6]}
    l_params= ['stride', 'padding' ,'optimizer','activation',
               'pooling','loss','kernel_size']
    i=0
    child=[]
    for params in l_params:
        l1 =len(parent_dict[params])
        prob=[round(r/(l1-1) ,4)]*l1
        value_parent=parent[i]
        i=i+1
        idx=parent_dict[params].index(value_parent)
        prob[idx]=1-r
        value_child=random.choices(parent_dict[params], prob , k=1)[0]
        child.append(value_child)
    return child
```

3.3 GET PARAMETERS FROM EA

This module is responsible for retrieving the neural network parameters to generate a working neural network architecture.

ALGORITHM OVERVIEW

INPUT: Parent gene sequence (model parameter) and 'init' parameter which is by default set as True. This parameter set as true indicates that it will initialize a parent gene sequence, when it is set false, it will mutate over the parent gene sequence passed as a parameter.
OUTPUT: Model architecture built based on the gene sequence generated by the EA.
Model ← build a model based on the gene sequence of model parameters
Return model.



```python
def get_params_fromEA(init=True, parent=None):
    if(init ==True):
        EA_params=EA_parent_init()
    else: #mutating over parent
        EA_params=mutation(parent)
    parent=EA_params
    stride,padding=EA_params[0],EA_params[1] #this will come EA
    opt ,act, pool,loss,kernel_size=EA_params[2],EA_params[3],EA_pa
    DL_params_dict={
        'cnn1':nn.Conv2d(3, 6, kernel_size=kernel_size, padding=pad
        'cnn2':nn.Conv2d(6, 16, kernel_size=kernel_size),
        'cnn3':nn.Conv2d(16, 120, kernel_size=kernel_size),
        'cnn4':nn.Conv2d(120, 84, 1),
        'act_sigm': nn.Sigmoid(),
        'act_relu':nn.ReLU(),
        'pool_max':nn.MaxPool2d(2, stride=stride),
        'pool_avg':nn.AvgPool2d(2),
        'cnn_last':nn.Conv2d(84, 10, 1),
        'loss_cross':nn.CrossEntropyLoss(),
        'loss_nll':nn.NLLLoss()
    }
    lenet = nn.Sequential(
    DL_params_dict['cnn1'],
    DL_params_dict[act],
    DL_params_dict[pool],
    DL_params_dict['cnn2'],
    DL_params_dict[act],
    DL_params_dict[pool],
    DL_params_dict['cnn3'],
    DL_params_dict[act],
    DL_params_dict['cnn4'],
```

3.4 TRAIN DL

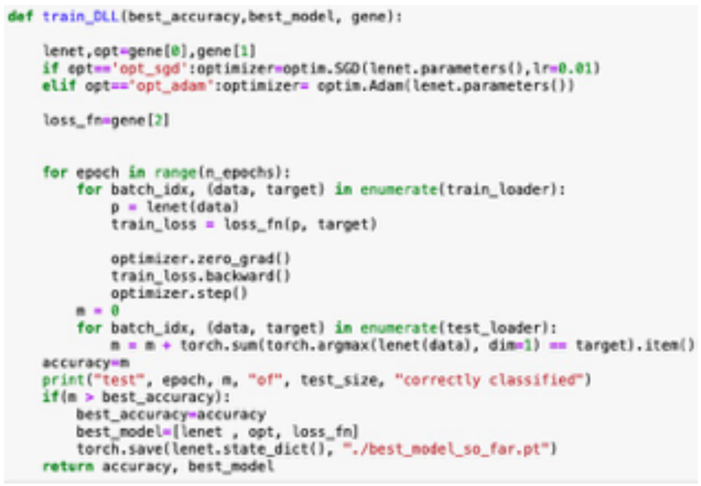
This module is responsible for training the DL model based on the model received from above function. It keeps a track of best accuracy seen so far and best model seen so far. It saves any model that outperforms previously best seen model to the memory.

It trains the DL model for n_epochs fixed in the beginning of the code. It tests the model on a test set of 10k unseen instances. It calculates the validation accuracy and uses it to compare the models seen so far.

ALGORITHM OVERVIEW

INPUT: Model architecture as per the gene sequence, the best accuracy's seen so far, the model that gave the best accuracy.
OUTPUT: Accuracy of the current model and best model so far after the current iteration.

Accuracy <-- The model accuracy after training for *n_epohcs* and testing over validation set.
Return accuracy.

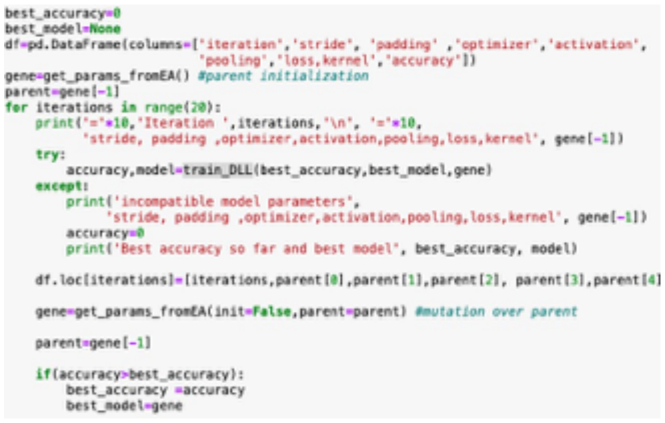


```
def train_DLL(best_accuracy,best_model, gene):

    lenet,opt=gene[0],gene[1]
    if opt=='opt_sgd':optimizer=optim.SGD(lenet.parameters(),lr=0.01)
    elif opt=='opt_adam':optimizer= optim.Adam(lenet.parameters())

    loss_fn=gene[2]


    for epoch in range(n_epochs):
        for batch_idx, (data, target) in enumerate(train_loader):
            p = lenet(data)
            train_loss = loss_fn(p, target)

            optimizer.zero_grad()
            train_loss.backward()
            optimizer.step()
        m = 0
        for batch_idx, (data, target) in enumerate(test_loader):
            m = m + torch.sum(torch.argmax(lenet(data), dim=1) == target).item()
    accuracy=m
    print("test", epoch, m, "of", test_size, "correctly classified")
    if(m > best_accuracy):
        best_accuracy=accuracy
        best_model=[lenet , opt, loss_fn]
        torch.save(lenet.state_dict(), "./best_model_so_far.pt")
    return accuracy, best_model
```

3.5 DRIVER MODULE

This module is responsible for calling the above calling the above functions for 20 times, each time either generating a random gene sequence by calling *get_params_from_EA(),* keeping *‘init’*=True or calling it to generate a mutated gene sequence of ‘parent’ gene sequence, by keeping *‘init’*= False. It uses the *gene* sequence returned by *get_params_from_EA() to* train the DL model by calling *train_DL()*. It compares the accuracy of model with all other models trained so far, and stores the best model.
It also keeps a record of all the iterations of EA in the pandas dataframe *df.* This is used later to analyze the important relationship between the model parameters and performance of the model.

ALGORITHM OVERVIEW

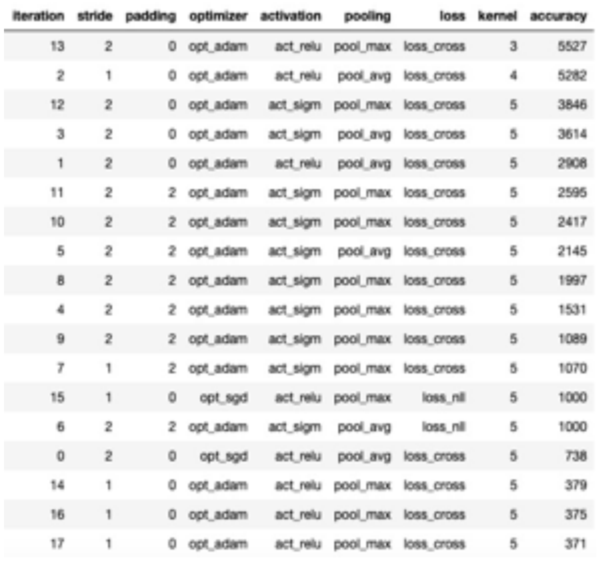


```
best_accuracy=0
best_model=None
df=pd.DataFrame(columns=['iteration','stride', 'padding' ,'optimizer','activation',
                         'pooling','loss,kernel','accuracy'])
gene=get_params_fromEA() #parent initialization
parent=gene[-1]
for iterations in range(20):
    print('='*10,'Iteration ',iterations,'\n', '='*10,
          'stride, padding ,optimizer,activation,pooling,loss,kernel', gene[-1])
    try:
        accuracy,model=train_DLL(best_accuracy,best_model,gene)
    except:
        print('incompatible model parameters',
              'stride, padding ,optimizer,activation,pooling,loss,kernel', gene[-1])
        accuracy=0
        print('Best accuracy so far and best model', best_accuracy, model)

    df.loc[iterations]=[iterations,parent[0],parent[1],parent[2], parent[3],parent[4]

    gene=get_params_fromEA(init=False,parent=parent) #mutation over parent

    parent=gene[-1]

    if(accuracy>best_accuracy):
        best_accuracy =accuracy
        best_model=gene
```

IV. RESULT

A log of the 20 iterations with different gene sequences shows interesting relationships between the model parameters and the performance of the model. Dataset used in CIFAR10 and model used is CNN.

•Stride -2, or 1; Padding-0 ; Activation –ReLu ; Pooling –avg or max and Kernel- 3 or 4 give the best model architecture.
•Activation sigmoid in hidden layers does not perform good.
•Kernel size>4 worsen the performance of model.
•Optimizer and Loss functions don't play much role in model performance.
•Both max and avg pooling give out similar kind of results.



iteration	stride	padding	optimizer	activation	pooling	loss	kernel	accuracy
13	2	0	opt_adam	act_relu	pool_max	loss_cross	3	5527
2	1	0	opt_adam	act_relu	pool_avg	loss_cross	4	5282
12	2	0	opt_adam	act_sigm	pool_max	loss_cross	5	3846
3	2	0	opt_adam	act_sigm	pool_avg	loss_cross	5	3614
1	2	0	opt_adam	act_relu	pool_avg	loss_cross	5	2908
11	2	2	opt_adam	act_sigm	pool_max	loss_cross	5	2595
10	2	2	opt_adam	act_sigm	pool_max	loss_cross	5	2417
5	2	2	opt_adam	act_sigm	pool_avg	loss_cross	5	2145
8	2	2	opt_adam	act_sigm	pool_max	loss_cross	5	1997
4	2	2	opt_adam	act_sigm	pool_max	loss_cross	5	1531
9	2	2	opt_adam	act_sigm	pool_max	loss_cross	5	1089
7	1	2	opt_adam	act_sigm	pool_max	loss_cross	5	1070
15	1	0	opt_sgd	act_relu	pool_max	loss_nll	5	1000
6	2	2	opt_adam	act_sigm	pool_avg	loss_nll	5	1000
0	2	0	opt_sgd	act_relu	pool_avg	loss_cross	5	738
14	1	0	opt_adam	act_relu	pool_max	loss_cross	5	379
16	1	0	opt_adam	act_relu	pool_max	loss_cross	5	375
17	1	0	opt_adam	act_relu	pool_max	loss_cross	5	371

V. CONCLUSION

The best model determined by Evolutionary Algorithm in just 20 iterations gave a convolutional neural network model that predicted ~4000/10000 correct predictions without preprocessing the data. Whereas, when the same data is trained and tested on LeNet Model architecture which is one of the famous benchmark Neural network architecture it predicted 5000/10000 over validation data. LeNet model parameters are developed through months of research. With the given amount of time and resources, Evolutionary algorithm can only be run on 20 iterations on a local machine. As per our estimation, greater number of iterations of Evolutionary Algorithm may give same results as LeNet or might even outperform LeNet.

VI. BIBLIOGRAPHY

1. PhamTA,TranVQ,VuH-LT,LyH-B(2020)**Designdeepneuralnetwork architecture using a genetic algorithm for estimation of pile bearing capacity.** PLoS ONE 15(12): e0243030. https://doi.org/10.1371/ journal.pone.0243030
2. Islam, Dr-Badar & Baharudin, Zuhairi & Raza, Muhammad Qamar & Nallagownden, Perumal. (2014). **"Optimization of neural network architecture using genetic algorithm for load forecasting."** 1-6. 10.1109/ ICIAS.2014.6869528.
3. Albadr MAA, Tiun S, Ayob M, AL-Dhief FT, Omar K, Hamzah FA (2020), **"Optimised genetic algorithm-extreme learning machine approach for automatic COVID-19 detection."** PLoS ONE 15(12): e0242899. https:// doi.org/10.1371/journal.pone.0242899