CSL7540 - Course Project Report

1st Aarti Pol (M20MA002)
Mathematics Department
IIT Jodhpur

2nd Rohith Jade (M21EE006)
Electrical Department
IIT Jodhpur

3rd Vikram Reddy Ettam (M21EE011)
Electrical Department
IIT Jodhpur

***Abstract*—In this project, we worked on a search algorithm to do Neural Architecture Search(NAS) for finding the best performing Convolution Neural Network(CNN) architecture on the fashion-mnist dataset. We used a Genetic Algorithm to find the best architecture. In Genetic Algorithm, we start with a random population and successor states are generated by combining the parents from previous generation. These parents are selected based on their fitness. We combined test accuracy and the number of parameters to create our fitness function. Eventually, we were able to search the CNN model over multiple generations using the algorithm. The best performing model (genome) in terms of fitness function gave the test accuracy of 80% with around $3.2k$ parameters while the model with the best accuracy achieved score of 85% with $8.7k$ parameters.**

I. Dataset

We are using fashion-minist dataset which has $60k$ train and $10k$ test samples. There are 10 classes in the dataset - *'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'*. These classes are equally distributed across train and test samples.

II. Model

The base model for our NAS is Convolution Neural Network which performs convolution operation on an input image. During a convolution operation, a kernel/filter is applied on the input image to get an output image of smaller or similar dimension. In CNN, there are multiple convolution layers applied one after another which helps in extracting high-level features from images.In this project, we have some constraints on the CNN architecture we can use:

- For Normal CNN (NC) layers: *stride=1, padding=same, 1<=kernel size<8*
- For Reduction CNN (RC) layers: *stride=2, padding=valid*
- Final layers are fixed: *Global Average Pooling 2D layer, Fully Connected/Dense. layer with 64 units, Fully Connected/Dense layer with 10 units*

In terms of activation functions we can use *relu, sigmoid, tanh, swish, gelu* for NC and RC layers but the all final layers must use same activation. The CNN architecture is represented using a genome string using the format: *<NC or RC> <no. of CNN filters> <kernelsize><activation function>;FL <activation function>*.

III. Algorithm

We used Genetic Algorithm to find best neural architecture while maintaining the constraints of the problem. Fig. 1 gives an illustration of genetic algorithm over a population of four samples.

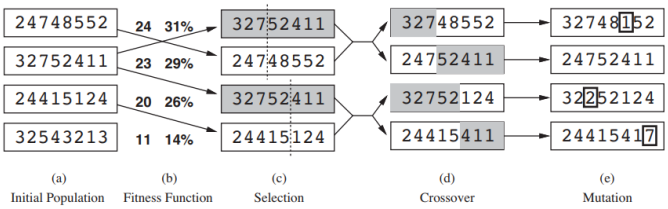

Fig. 1. Genetic algorithm for 4 sample strings

We perform the search strategy in the following main steps:

1. We generate random genome strings of $max\_population\_size$ that meet the mentioned constraints. This initial set is called the population.

2. We train the model represented by these genome strings and store their test accuracy and parameter count.

3. After this, we calculate the fitness score of each individual genome in the population. We designed a fitness function to take both test accuracy and parameter count into consideration. Overall, we want to find an architecture such that the test accuracy value is high and the parameter count is low. We used the following fitness function to achieve this objective:

$$fitness\_score = \frac{accuracy}{log(numberofparameters)}$$

Scale of the test accuracy (up to 100) is quite different from parameter count (in 1000s), so we divide test accuracy by logarithm of parameter count so that overall fitness function doesn't get dominated only by parameter count.

4. Now in each generation, for reproduction, individuals are selected randomly according to the probabilities assigned by their fitness score. This ensures that the individual with better fitness score has more probability of getting selected for reproduction. In the crossover process, a crossover point is chosen at random and the child inherits the features of both parents.

5. After crossover, we perform random mutation on the individuals. A less probability is assigned to decide whether mutation has to take place or not because mutation is only needed occasionally and it helps to come out of local optimum point while exploring the search space. During mutation, we select a random point in genome string and replace it with another random element from possible set of all elements in population. This helps to bring a divergence element into the picture by keeping the genetic diversity.

6. We repeat points 2 to 5 for either a fixed number of generations or until a desired fitness score is achieved.

IV. Heuristics

As the search space in NAS is quite large, we applied some heuristics rules to reduce the space based on experiments on several models.

1. In order keep number of parameters low, we set number of normal convolution layer in the range [1, 2, 3] and number of filters in the range [2, 16, step=2]. For kernel size the search space is [3, 4, 5].

2. We also tried several optimizers while keeping the model architecture same. From these experiments, we observed that SGD optimizer with $learning\_rate = 0.01$, $momentum = 0.9$ was achieving good test accuracy in lesser number of epochs. Since, the project is computationally expensive because of searching over large number of networks, we decided to use this optimizer.

3. We have used $max\_population\_size = 25$, $number\_of\_generations = 4$, $epochs = 20$ while training the genetic algorithm. We ran short experiments with various values of these parameters but these are the final used values.

V. Additional Details

1. There are many ways randomness can occur while training the model - weight initialization, data shuffle etc.. In order to compare results across genomes properly, we set the seed to various methods so that experiments are run deterministically.

2. During the NAS, same genome could be generated multiple times due to crossover and mutation. In order to not train the model again on these genomes, we kept a log of all genomes and their score. This saved significant compute time.

3. In order to maintain constraints of the problem, we have to reject some mutations and crossover where number of RC layers and FL layers exceeded the mentioned counts.

VI. Results

In the following tables we the top 5 models based on 3 criteria. Figure 2 shows top models as per fitness functions defined above. We can see that top model get 80% accuracy with only around 3k parameters. Similarly, Figure 3 and Figure 4 shows top 5 models based on accuracy and fitness score including time which is defined as:

$$fitness\_score' = \frac{fitness\_score}{time\_taken\_to\_train}$$

Following are the genome strings for the best models:

- fitness: *NC 16 3 tanh;NC 6 3 swish;RC 6 3 tanh;RC 10 3 tanh;FL swish;* - achieves 80% test accuracy with $3.2k$ params.
- accuracy: *NC 16 4 gelu;RC 10 4 relu;NC 8 4 gelu;NC 16 4 tanh;RC 4 5 tanh;FL swish;* - achieves 85% test accuracy with $8.7k$ params.
- fitness(with time): *NC 10 4 relu;RC 14 4 tanh;RC 14 4 relu;FL tanh;* - achieves 75% test accuracy with $7.1k$ params but took only 66 seconds to train.
- manual inspection: *NC 14 3 relu;NC 8 3 gelu;RC 12 3 tanh;RC 12 4 gelu;FL gelu;* - achieves 85% test accuracy with $5.8k$ params. This model is $3rd$ as per fitness and $2nd$ as per accuracy.

We also calculated the results of best performing model (in terms of fitness) as shown in Figure 5. This model achieved test accuracy of 80% but from classification report, we can observe that it's not performing well across all classes. The f1 score of class 2, 4 and 6 are well below the overall f1 scores.

	genome	test_acc	train_acc	num_param	time_taken(sec)	fitness	fitness_time
35	NC 16 3 tanh;NC 6 3 swish;RC 6 3 tan...	0.8063	0.8064	3264	183.51	9.9658	0.0543
12	NC 14 3 relu;NC 8 3 gelu;RC 12 3 tan...	0.8184	0.8240	4476	159.94	9.7353	0.0609
31	NC 14 3 relu;NC 8 3 gelu;RC 12 3 tan...	0.8413	0.8428	5830	153.03	9.7027	0.0634
28	NC 10 4 swish;NC 6 3 swish;RC 6 3 ta...	0.7757	0.7763	3426	176.80	9.5305	0.0539
24	NC 16 3 tanh;NC 6 3 swish;RC 6 3 tan...	0.7822	0.7851	3740	172.07	9.5079	0.0553

Fig. 2. Best Genome as per fitness score

	genome	test_acc	train_acc	num_param	time_taken(sec)	fitness	fitness_time
29	NC 16 4 gelu;RC 10 4 relu;NC 8 4 gel...	0.8538	0.8563	8768	143.79	9.4043	0.0654
31	NC 14 3 relu;NC 8 3 gelu;RC 12 3 tan...	0.8413	0.8428	5830	153.03	9.7027	0.0634
6	NC 16 4 tanh;NC 12 4 tanh;NC 14 4 sw...	0.8236	0.8249	9736	163.85	8.9682	0.0547
12	NC 14 3 relu;NC 8 3 gelu;RC 12 3 tan...	0.8184	0.8240	4476	159.94	9.7353	0.0609
11	NC 4 5 gelu;NC 6 5 swish;RC 14 5 sig...	0.8223	0.8239	6504	144.66	9.3654	0.0647

Fig. 3. Best Genome as per test accuracy score

VII. Conclusions

In following diagrams, we can see the scatter plots of test_accuracy vs number of parameters. From the Figure 6 plot, we can see that the best models as per fitness scores are in top left corner shown by bright yellow color. In the Figure 7,

	genome	test_acc	train_acc	num_param	time_taken(sec)	fitness	fitness_time
23	NC 10 4 relu;RC 14 4 tanh;RC 14 4 re...	0.7548	0.7554	7184	66.05	8.5004	0.1287
2	NC 12 5 tanh;RC 10 5 sigmoid;RC 14 5...	0.7999	0.8049	8446	94.52	8.8470	0.0936
14	NC 6 5 relu;RC 14 5 sigmoid;RC 6 5 s...	0.7696	0.7746	5474	102.44	8.9408	0.0873
0	NC 8 4 sigmoid;RC 12 4 gelu;RC 12 4 ...	0.7831	0.7839	5482	113.01	9.0961	0.0805
25	NC 8 4 sigmoid;RC 12 4 gelu;RC 12 4 ...	0.7593	0.7616	5482	126.05	8.8196	0.0700

Fig. 4. Best Genome as per fitness with time score

	precision	recall	f1-score	support
0	0.67	0.86	0.75	1000
1	0.99	0.94	0.96	1000
2	0.79	0.58	0.66	1000
3	0.83	0.82	0.82	1000
4	0.66	0.67	0.67	1000
5	0.97	0.90	0.93	1000
6	0.48	0.48	0.48	1000
7	0.87	0.94	0.90	1000
8	0.90	0.94	0.92	1000
9	0.94	0.92	0.93	1000
accuracy			0.80	10000
macro avg	0.81	0.80	0.80	10000
weighted avg	0.81	0.80	0.80	10000

Fig. 5. Classification Report of Best Model (fitness)

the color scheme is based on fitness as well as time taken to train the model and we see that only few models are having bright colors which are in top center. It means that the the models in top left corner of Figure 6 were achieving higher accuracy scores with less number of parameters but they were taking more time to train. Hence, depending on the use case, we should consider all 3 parameters to pick the best model.

Also, as discussed in results section, another important consideration is use of metrics like f1 score to pick best models especially in multi-class classification setting where we want the model to be good across all classes.

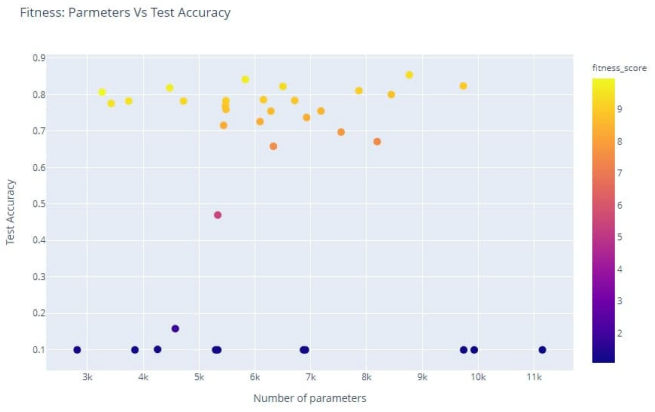

Fig. 6. Scatter Plot of Accuracy vs Params with Fitness in color

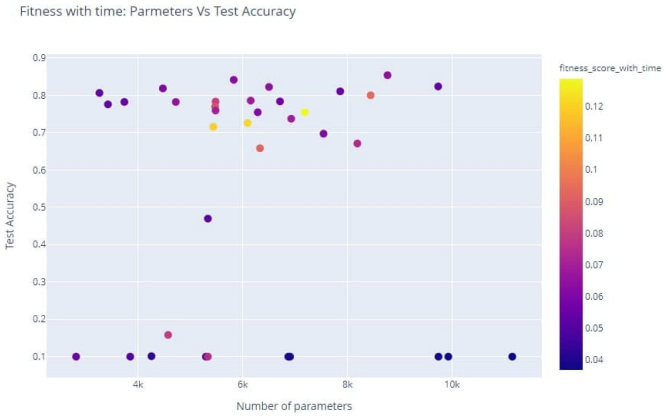

Fig. 7. Scatter Plot of Accuracy vs Params with Fitness(time) in color

REFERENCES

[1] Artificial Intelligence: A Modern Approach, Third Edition
[2] https://lilianweng.github.io/lil-log/2020/08/06/neural-architecture-search.html
[3] https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7